



US005353243A

United States Patent [19]

Read et al.

[11] Patent Number: **5,353,243**[45] Date of Patent: **Oct. 4, 1994****[54] HARDWARE MODELING SYSTEM AND METHOD OF USE**

[75] Inventors: **Andrew J. Read**, Sunnyvale; **Mark S. Papamarcos**; **Wayne P. Heideman**, both of San Jose; **Robert K. Mardjuki**, Peasanton; **Robert K. Couch**, Santa Cruz; **Peter R. Jaeger**; **William F. Kappauf**, both of San Jose; **Lawrence C. Widdoes, Jr.**; **Louis K. Scheffer**, both of Los Altos Hills, all of Calif.

[73] Assignee: **Synopsys Inc.**, Mountain View, Calif.

[21] Appl. No.: **939,393**

[22] Filed: **Aug. 31, 1992**

Related U.S. Application Data

[63] Continuation of Ser. No. 359,711, May 31, 1989, abandoned.

[51] Int. Cl.⁵ **G06F 15/20**

[52] U.S. Cl. **364/578**

[58] Field of Search **364/578, 579, 488, 490; 371/23, 24, 20.1, 27, 29.1; 324/73.1, 158 R**

[56] References Cited**U.S. PATENT DOCUMENTS**

3,720,131	3/1973	Frohock, Jr. et al.	324/158 R
3,764,995	10/1973	Holf, Jr. et al.	324/73.1
3,832,535	8/1974	DeVito 235/153 AC	
3,854,125	12/1974	Ehling et al.	340/172.5
3,922,537	11/1975	Jackson 235/153 AC	
3,976,940	8/1976	Chau et al.	324/73.1
4,055,801	10/1977	Pike et al.	324/73.1
4,092,589	5/1978	Chau et al.	324/73.1
4,102,491	7/1978	DeVito et al.	235/302
4,168,796	9/1979	Fulks et al.	324/73.1
4,216,533	8/1980	Ichimiya et al.	365/230
4,236,246	11/1980	Skilling 324/73.1	
4,450,560	5/1984	Conner 371/25.1	
4,456,880	6/1984	Warner et al.	324/158 R
4,488,354	12/1984	Chan et al.	29/830

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

150260A2 8/1984 European Pat. Off. .
2123156 1/1984 United Kingdom .
2164768 3/1986 United Kingdom G06F 15/60

OTHER PUBLICATIONS

L. C. Widdoes, Jr., and W. Harding, "CAE Station Uses Real Chips to Simulate VLSI-Based Systems," *Electronic Design*, Mar. 22, 1984, pp. 167-176.

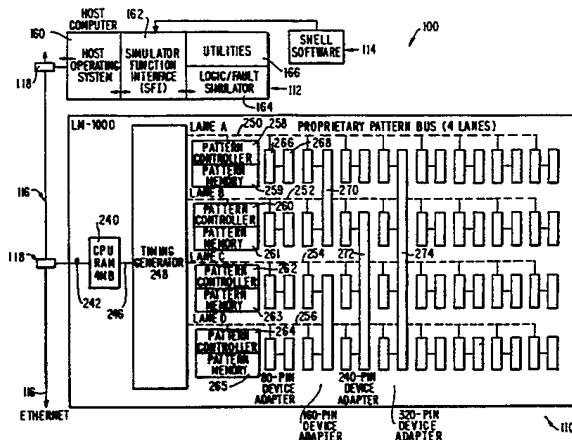
(List continued on next page.)

Primary Examiner—Ellis B. Ramirez

Attorney, Agent, or Firm—Marie H. MacNichol

[57] ABSTRACT

An improved hardware modeling system that is preferably embodied as a stand-alone system for networked connection to one or a variety of host computers that are used to design digital electronics systems, the hardware modeling system having a network interface for communicating between the hardware modeling system and the host computer, a central processing unit for controlling operation of the hardware modeling system, a central timing unit for generating timing signals for use in the operation of the hardware modeling system including the generation of precision clocks, data formatting strobes and sample strobes, an internal pattern bus for transmission of read/write requests from the central processing unit in one operational mode and pattern sequences for stimulation of the hardware modeling element in a second operational mode, a pattern controller for controlling presentation and delivery of the pattern sequences to the pattern bus, a pattern memory connected to the pattern controller for storing stimulus pattern sequences, pin electronics circuitry which is used for driving the pattern sequences on the pattern bus to the hardware modeling element and then sensing the five state values of the hardware modeling element pins, and an adapter that is used for fixturing the hardware modeling element to the pin electronics circuitry with the adapter supporting live insertion into a powered hardware modeling system.

111 Claims, 48 Drawing Sheets

U.S. PATENT DOCUMENTS

4,517,661	5/1985	Graf et al.	364/900
4,527,249	7/1985	Van Brunt	364/578
4,544,882	10/1985	Flora	324/73 R
4,587,625	5/1986	Marino, Jr. et al.	364/578
4,590,581	5/1986	Widdoes	364/578
4,594,544	6/1986	Necoechea	324/73
4,628,471	12/1986	Schuler et al.	364/578
4,635,218	1/1987	Widdoes	364/578
4,635,259	1/1987	Schinabeck et al.	364/579
4,639,664	1/1987	Chiu et al.	324/73
4,644,487	2/1987	Smith	364/578
4,646,299	2/1987	Schinabeck et al.	324/73.1
4,656,632	4/1987	Jackson	371/20
4,675,673	6/1987	Oliver	340/825.87
4,686,627	8/1987	Donovan et al.	364/481
4,695,968	9/1987	Sullivan, II et al.	364/578
4,714,875	12/1987	Balcoy et al.	324/73.1
4,724,378	2/1988	Murray et al.	324/73
4,744,084	5/1988	Beck et al.	371/23
4,764,925	8/1988	Grimes et al.	371/20
4,771,428	9/1988	Acuff	371/25
4,782,440	11/1988	Nominzu et al.	364/200
4,787,061	11/1988	Nei et al.	364/900
4,806,852	2/1989	Swan et al.	324/73 R
4,816,750	3/1989	Van Der Kloot et al.	324/73.1
4,931,723	6/1990	Jeffrey et al.	324/73.1

OTHER PUBLICATIONS

Stoll, "PMX: A Hardware Solution to the VLSI Model Availability Problem," IEEE Proceedings of the International Conference on Computer Design, Oct., 1985.

Johnson, "Considerations in Selecting a Physical Mod-

eling System," ADEE Technical Session Proceedings, Sep., 1986, pp. 225-231.

Johnson, "Software vs. Hardware Models for Simulation," Design Automation Guide, 1988.

S. Bisset, "LSI Tester Gets Microprocessors to Generate Their Own Test Patterns," Electronics, May 25, 1978, pp. 141-145.

Albrow, "2-head Auto-test System Takes on Complex VLSI," *Electronic Design*, Mar. 5, 1981, pp. 79-84.

Giles and Bowden, "Maintaining Simulation Accuracy Through Physical Device Models," IEEE Proceedings of the International Test Conference, 1985.

Parker, *Integrating Design and Test: Using CAE Tools for ATE Programming*, Computer Society Press of the IEEE, 1987.

Widdoes, L. C. Jr., and H. Stump, "Hardware Modeling," *VLSI Systems Design*, Jul., 1988.

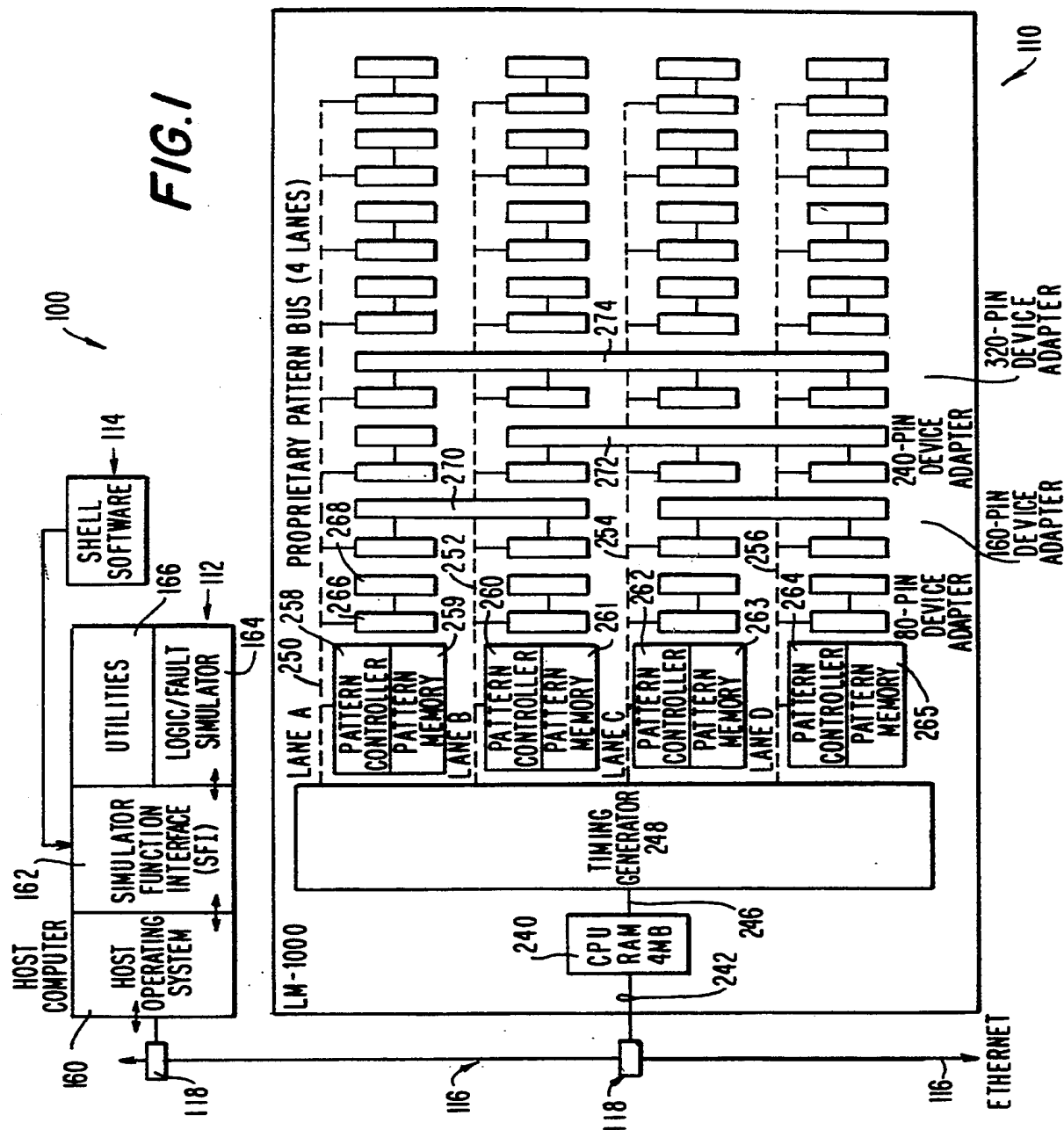
Gillette, "Tester Takes on VLSI With 264-K Vectors Behind Its Pins," Electronics, Nov. 3, 1981, pp. 122-127.

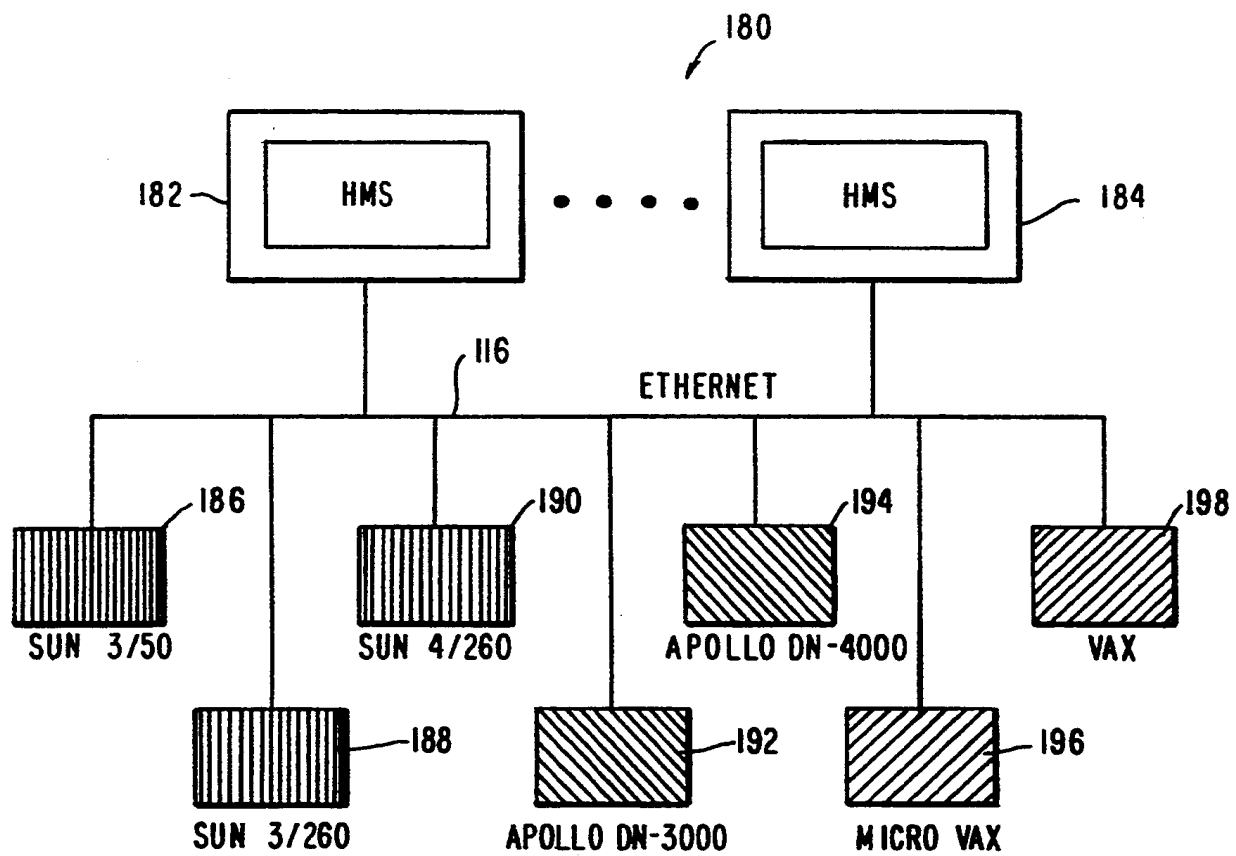
"Testing a TV Character Generator with the Sentry II Sequence Processor," *Fairchild Systems Technology*, pp. 1-12.

Fairchild Technical Bulletin 4, Nov. 1974.

Huston, R., "Description of the Intel 8085 Microprocessor Test Programs for the Sentry II/VII with Sequence Processor Module," *Fairchild Systems Technology*, Nov., 1977, pp. 1-12.

U.S. patent application Ser. No. 518,134, filed Oct. 25, 1974.



**FIG.2**

200

202	BEGIN/END	INITIALIZES AND TERMINATES AN HMS MODELING SESSION.
204	CREATE	CREATES VARIOUS HMS ENTITIES.
206	RELEASE	RELEASES (FREES) VARIOUS HMS ENTITIES (COMPLEMENT OF CREATE).
208	SET	SETS PARAMETERS AND DEVICE PIN VALUES.
210	GET	RETRIEVES PARAMETERS AND DEVICE PIN VALUES (COMPLEMENT OF SET).
212	SAVE/RESTORE	SAVES/RESTORES THE FULL HMS STATE OF A SIMULATION.
214	INQUIRE	QUERIES THE STATE OF VARIOUS HMS PARAMETERS AND USAGE INFORMATION.
216	MISCELLANEOUS	PROVIDES VARIOUS UTILITIES.

FIG. 3

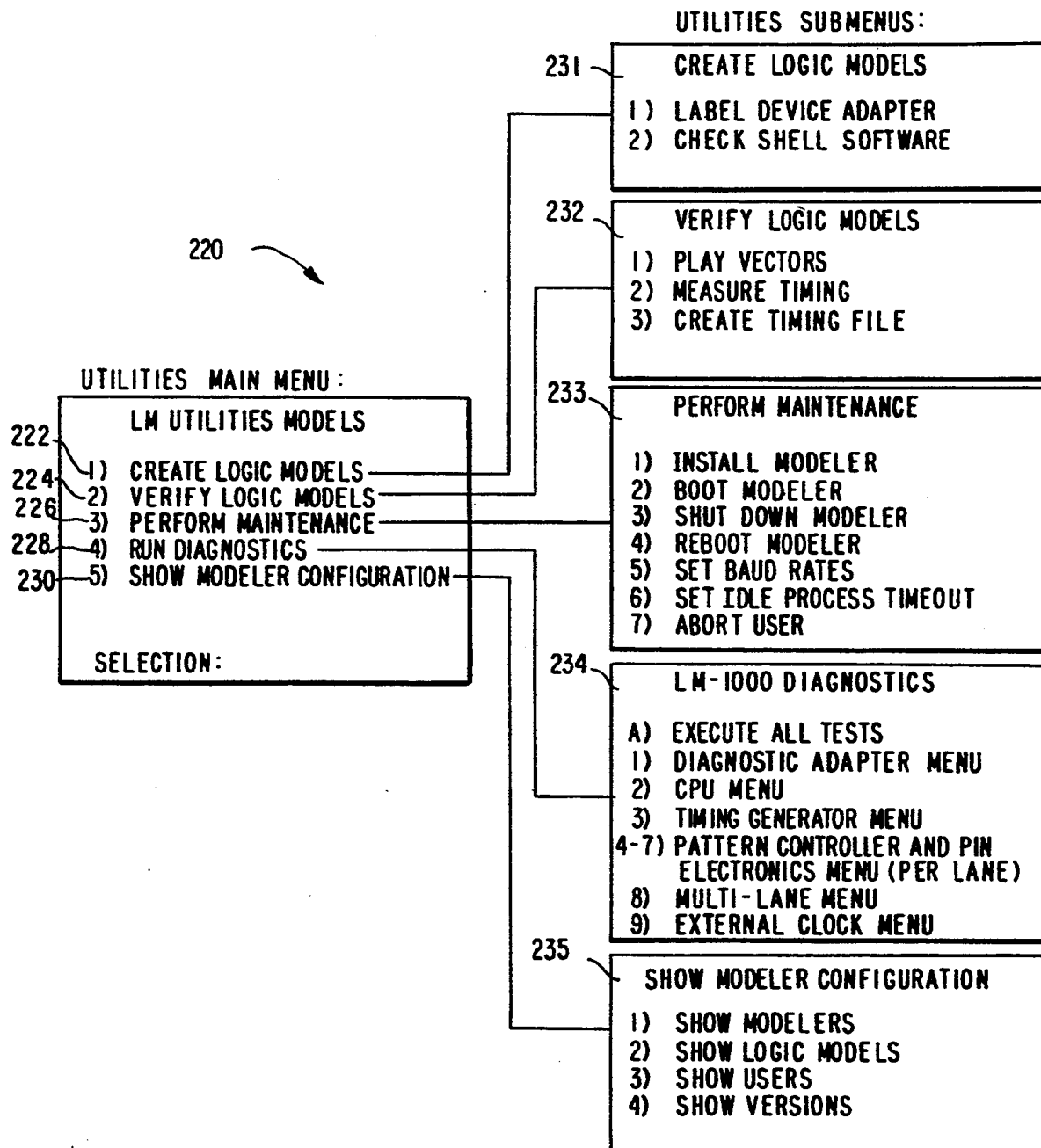
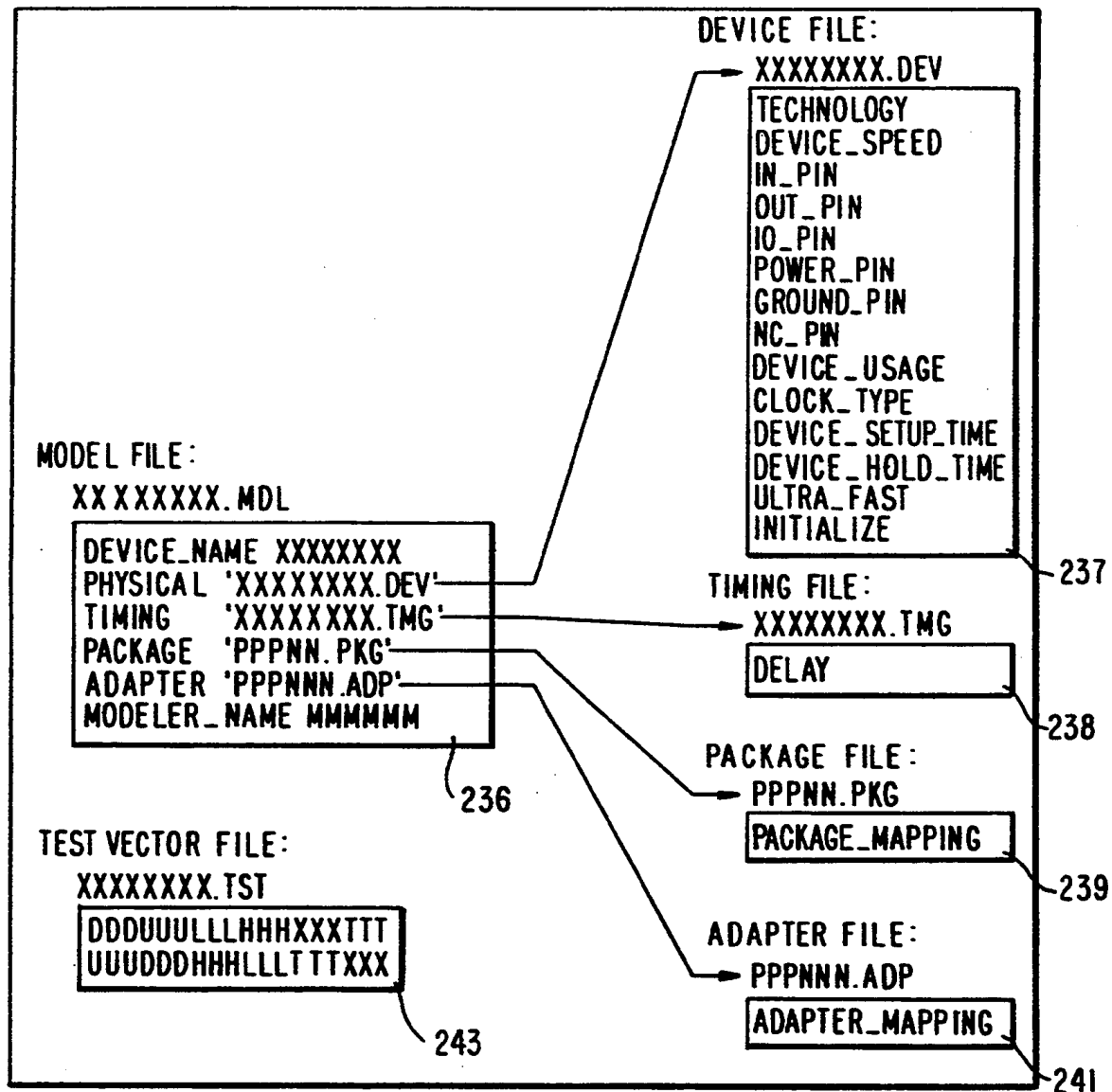


FIG.4

**FIG.5**

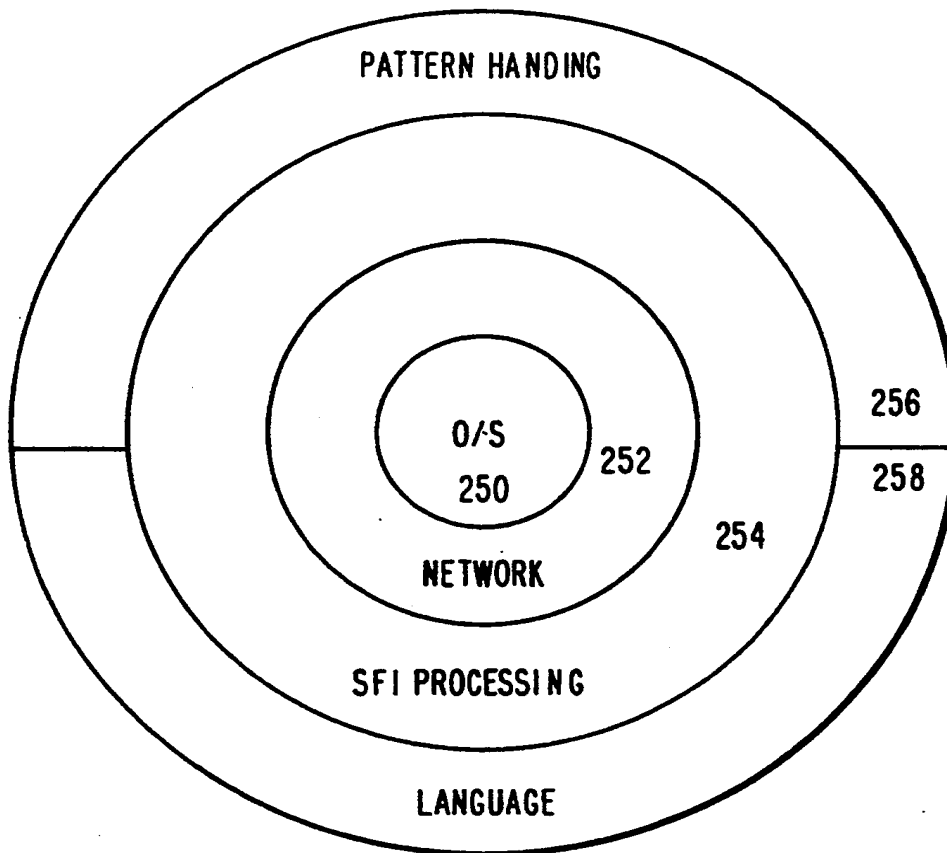
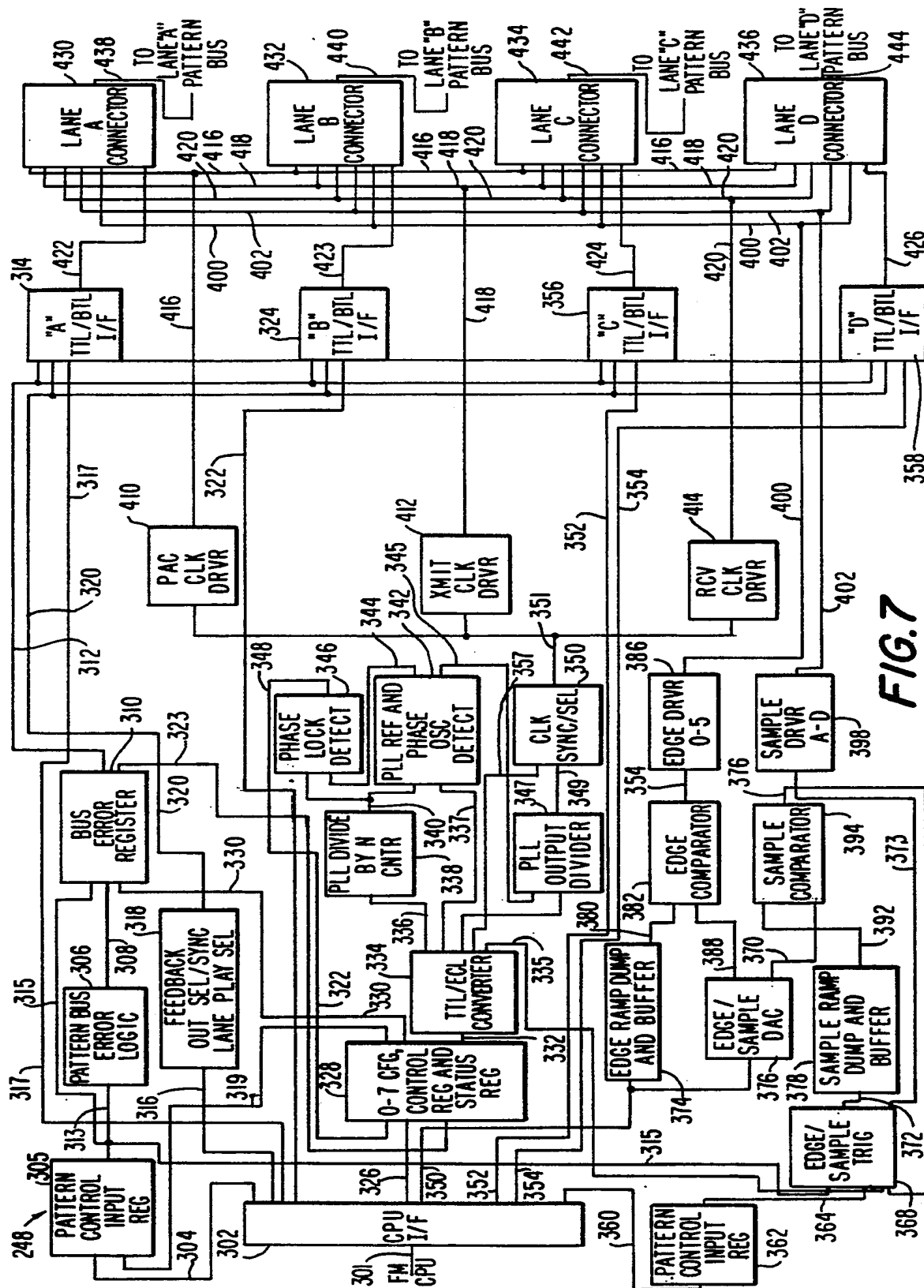
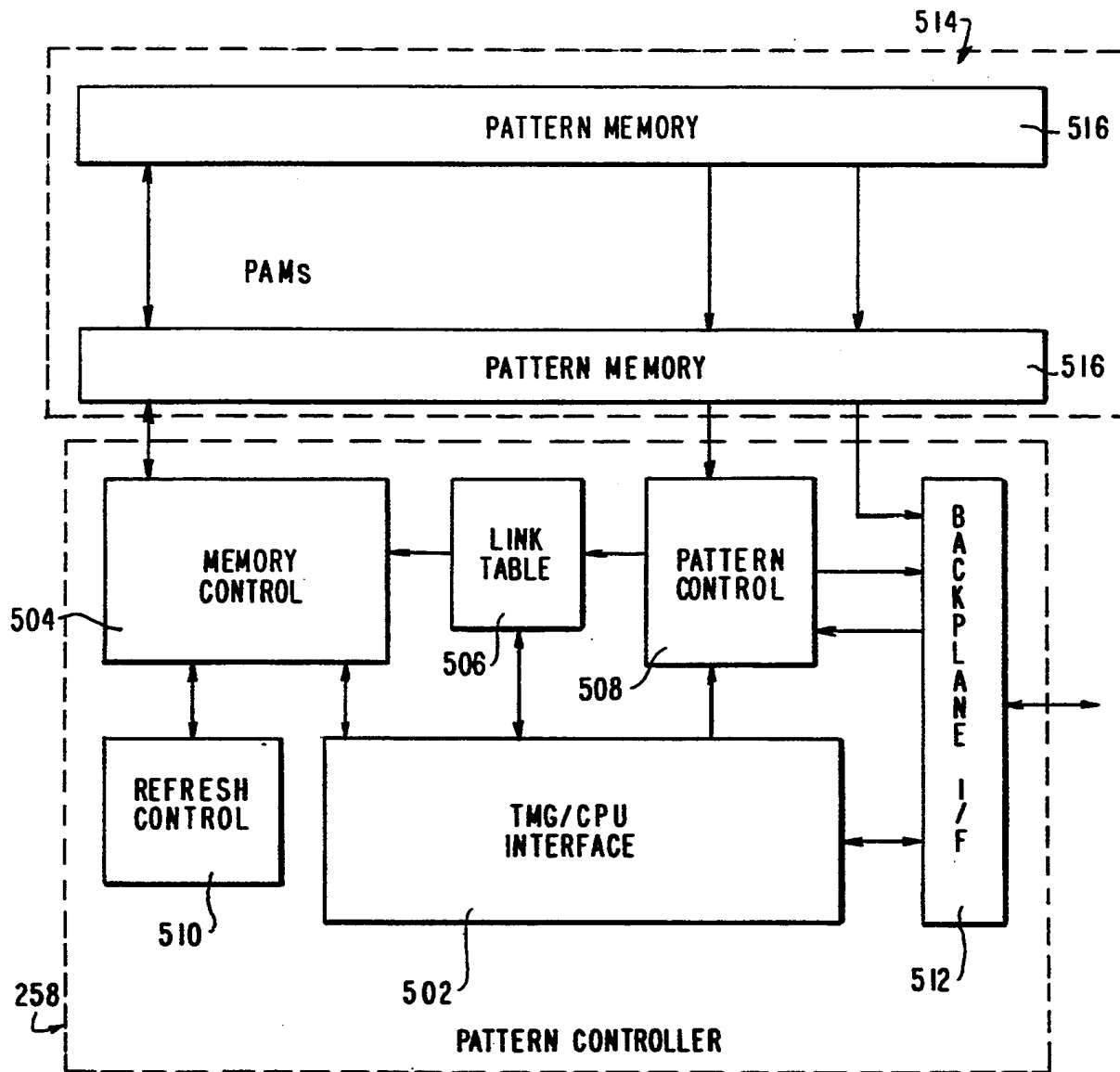


FIG. 6



**FIG. 8**

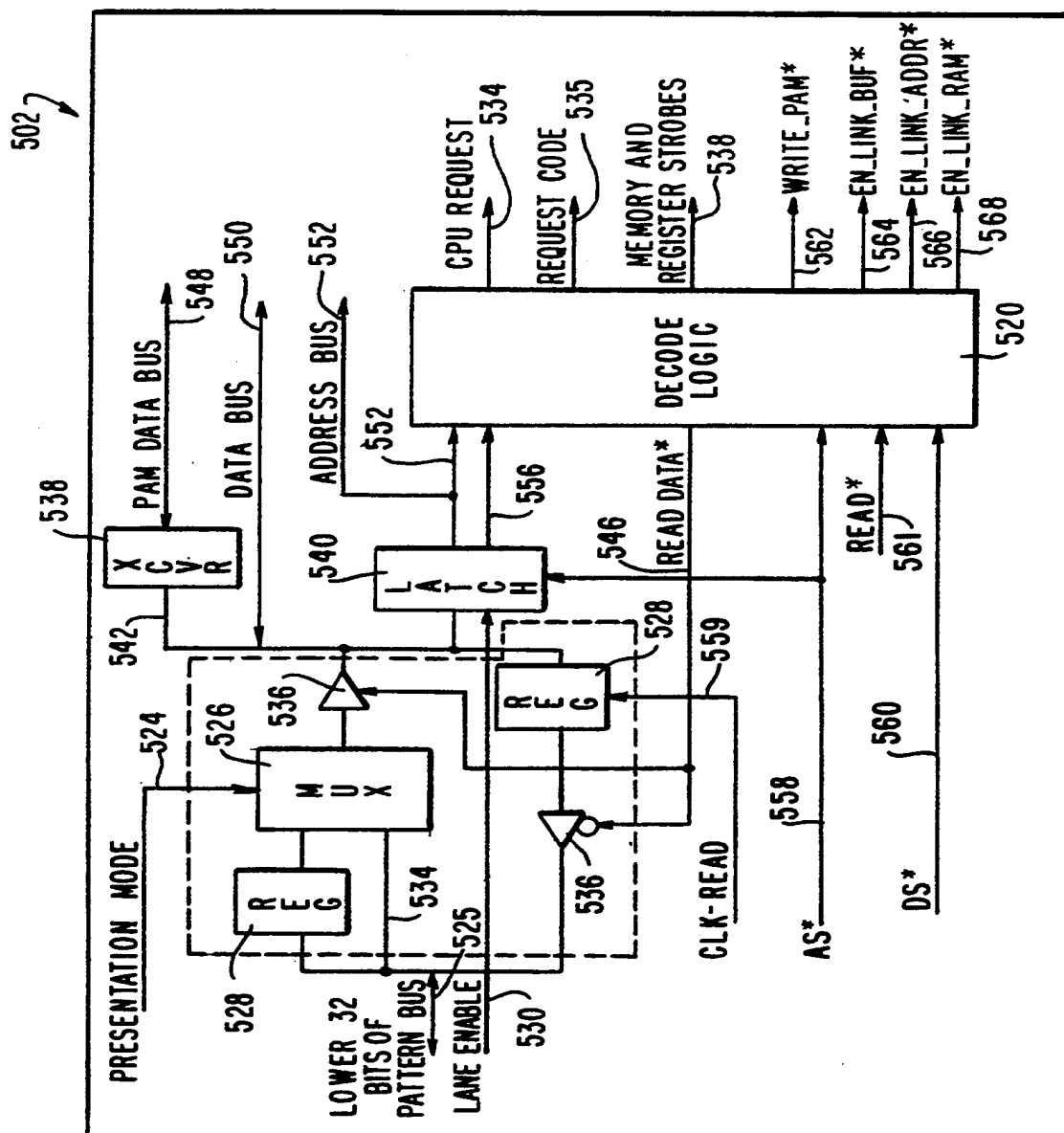
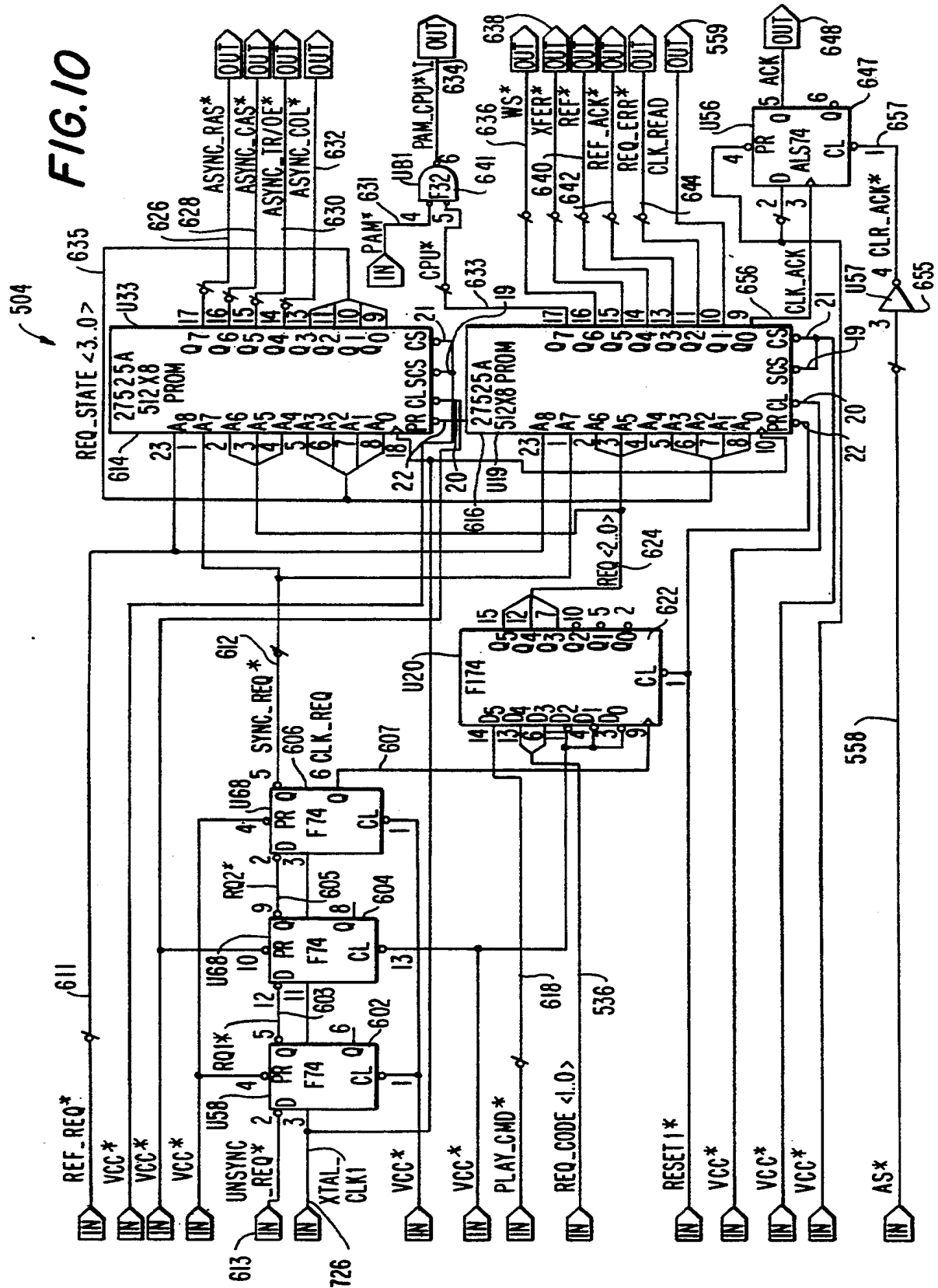
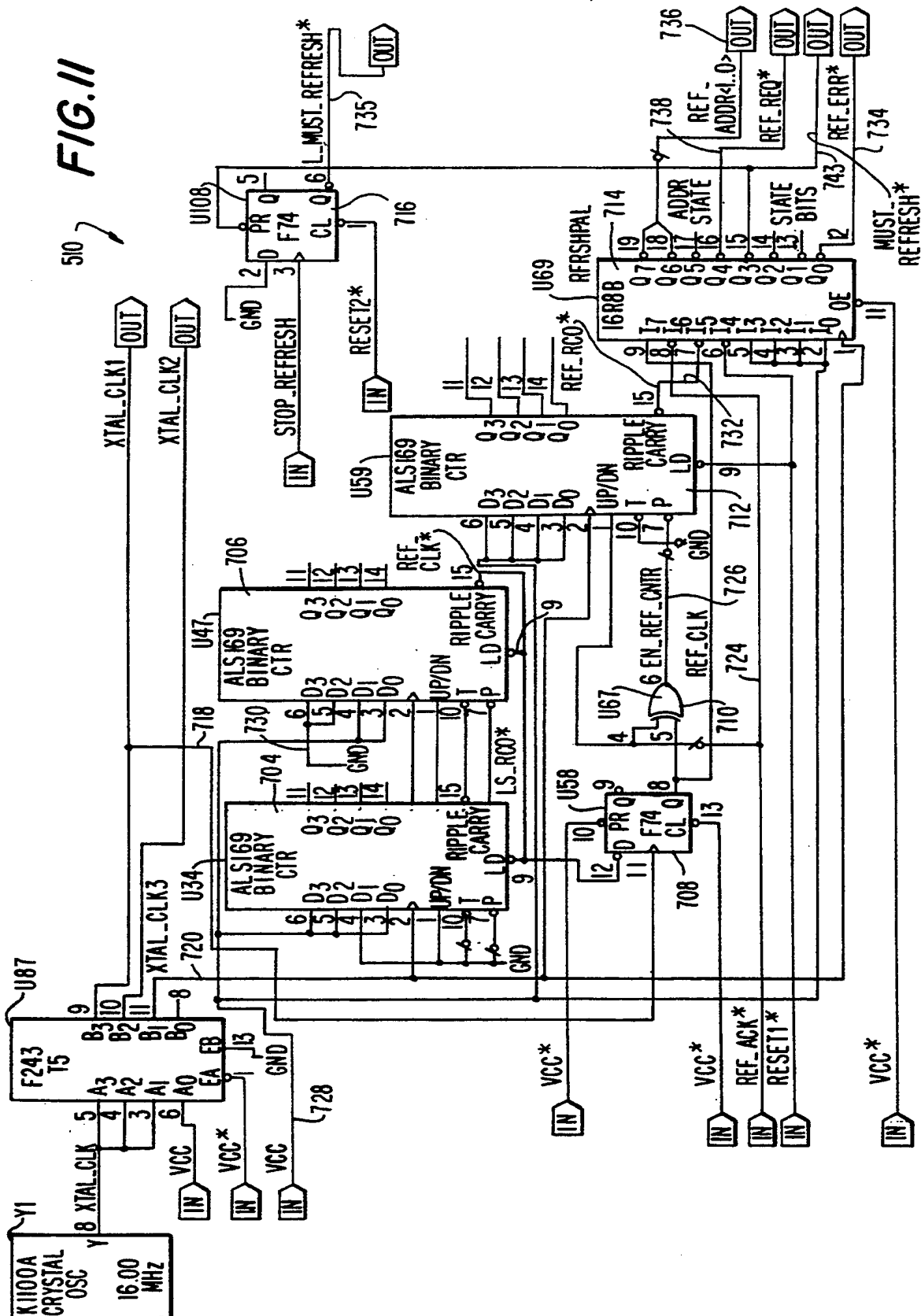
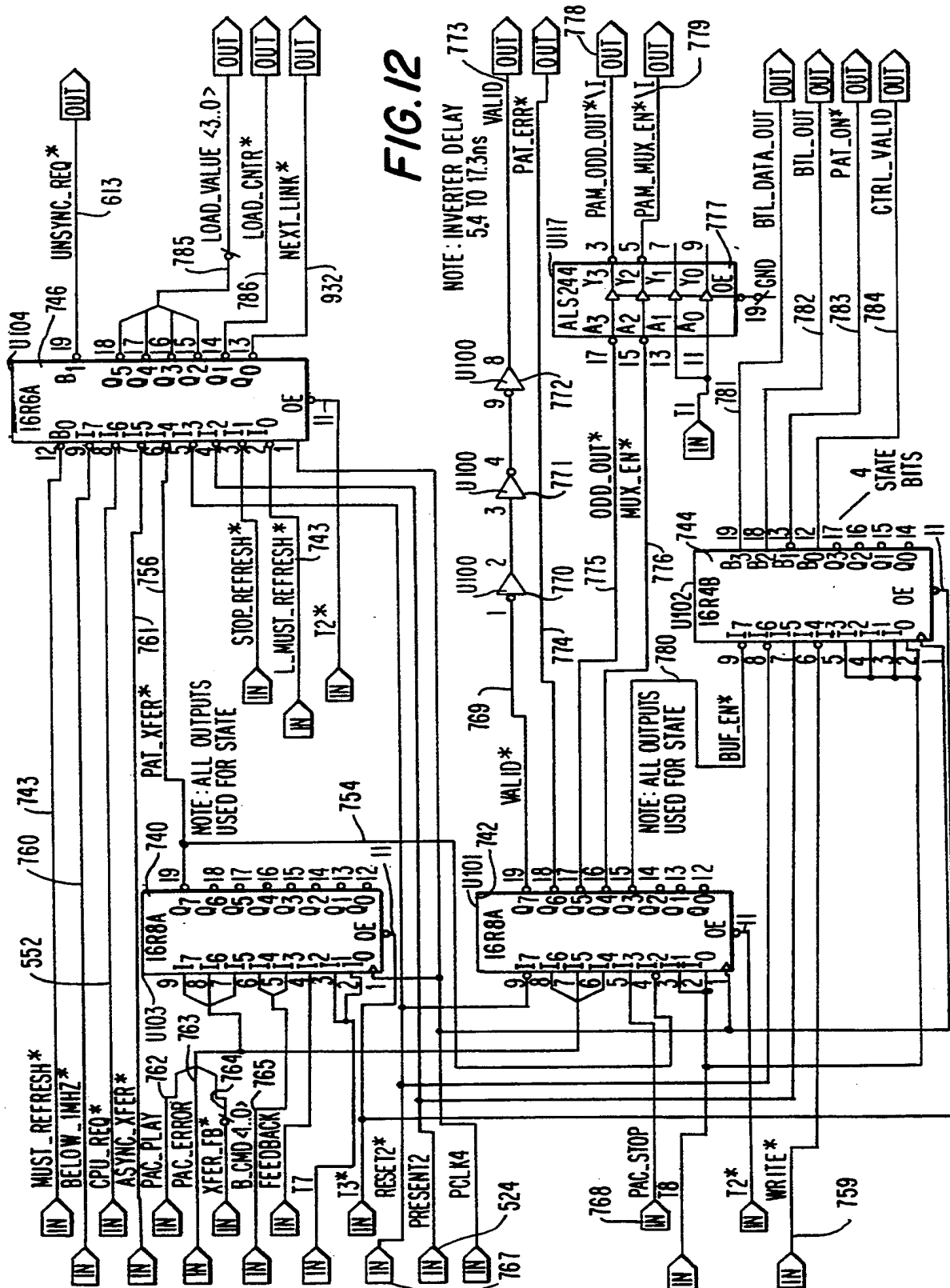


FIG. 9







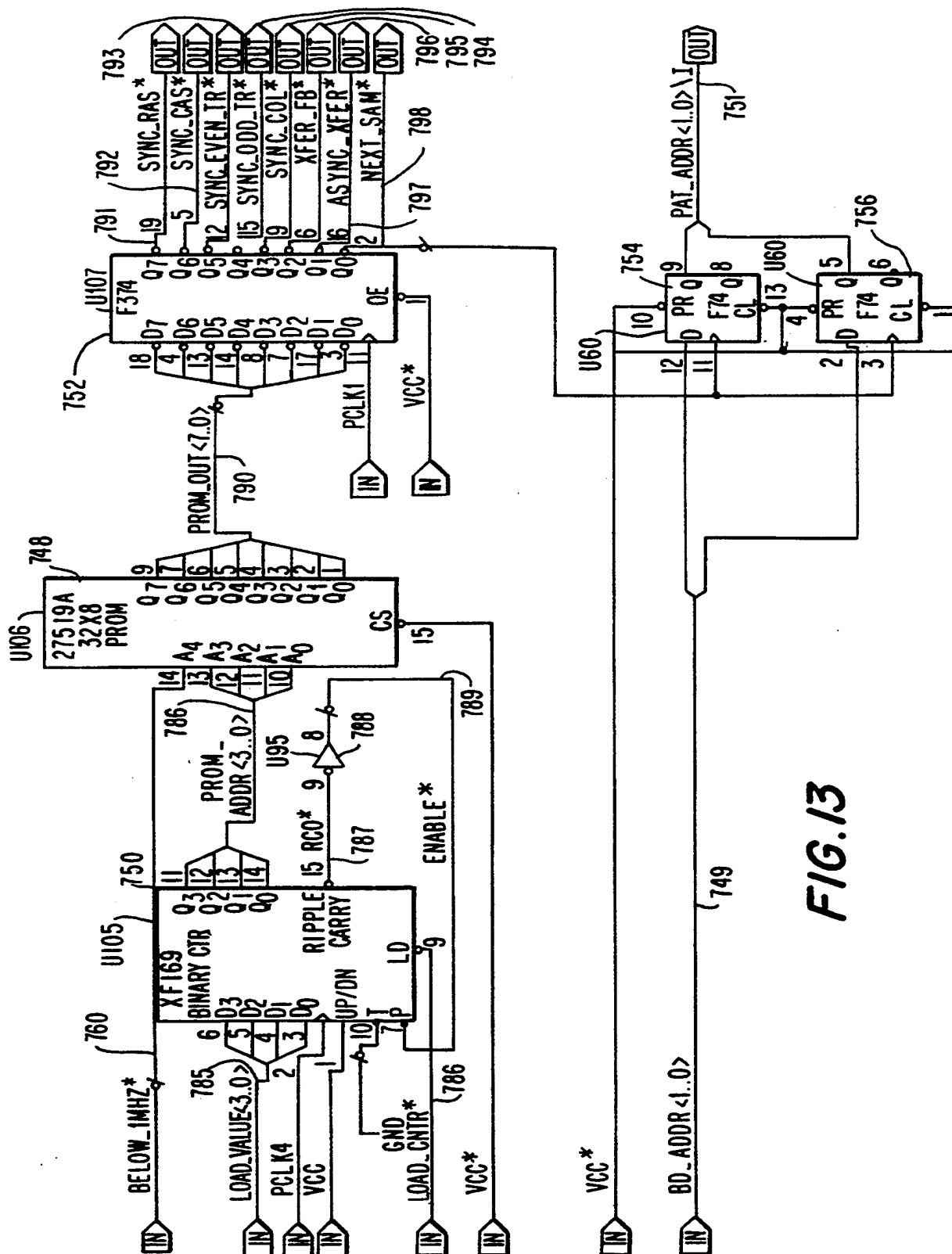


FIG. 13

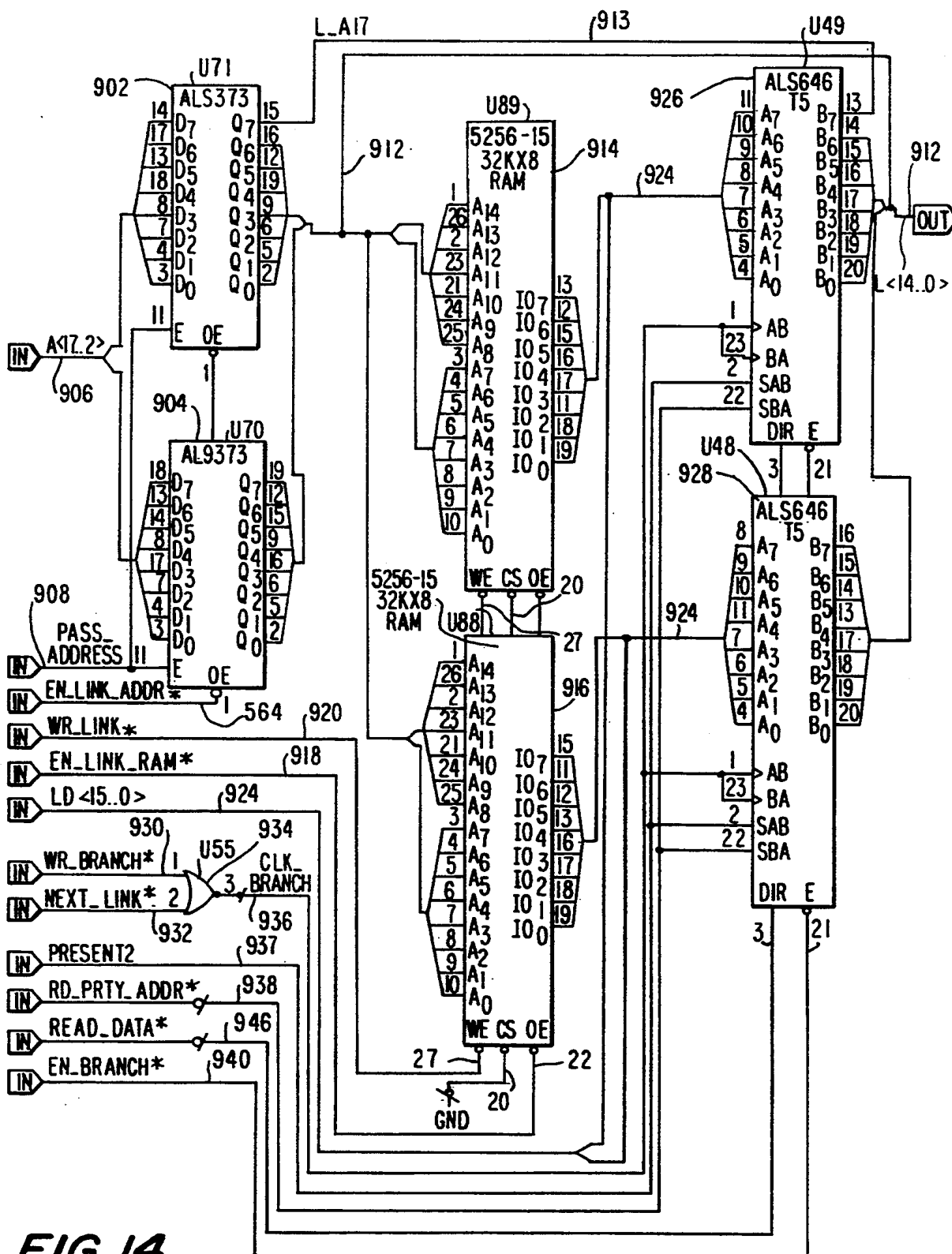


FIG. 14

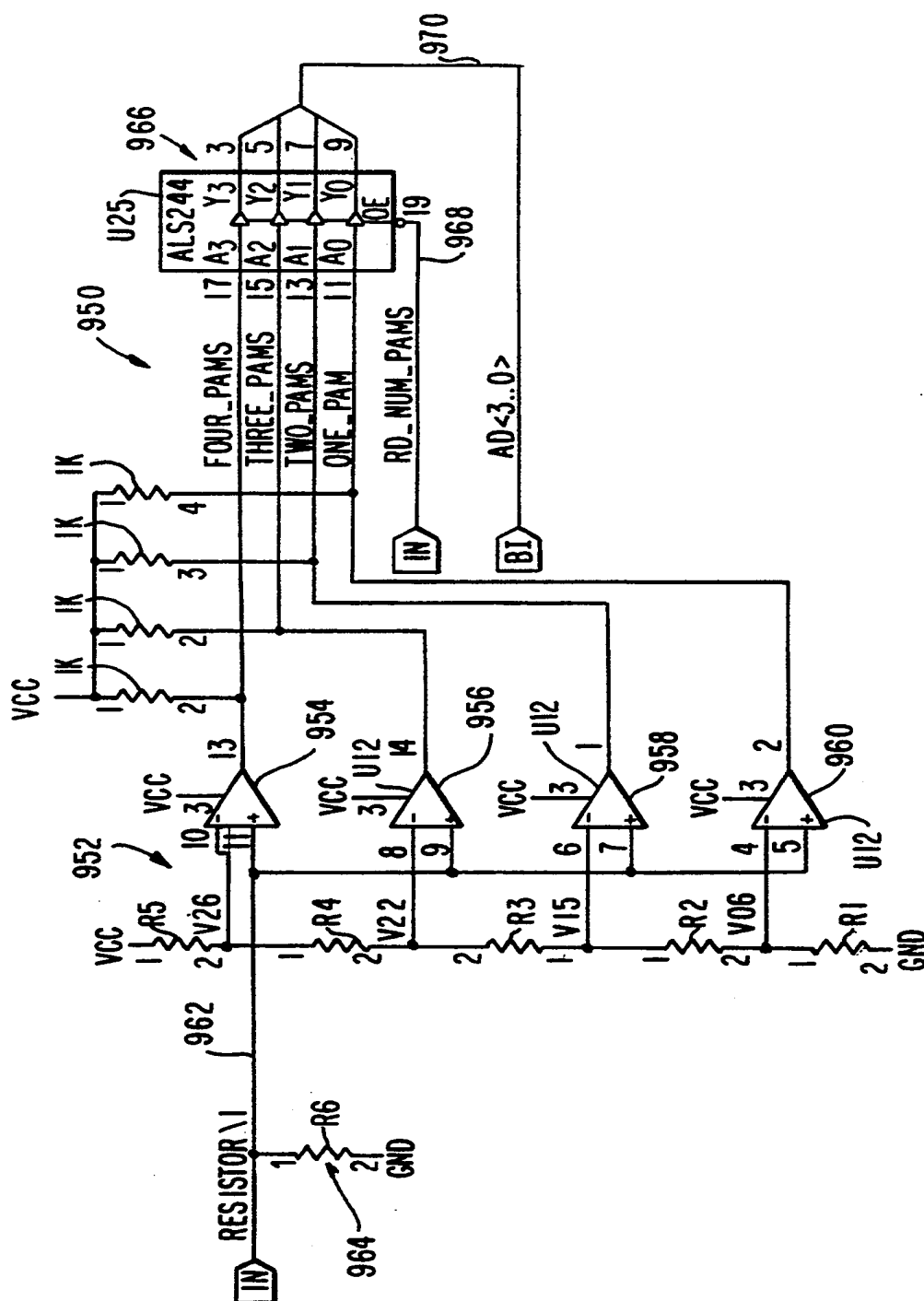


FIG. 15

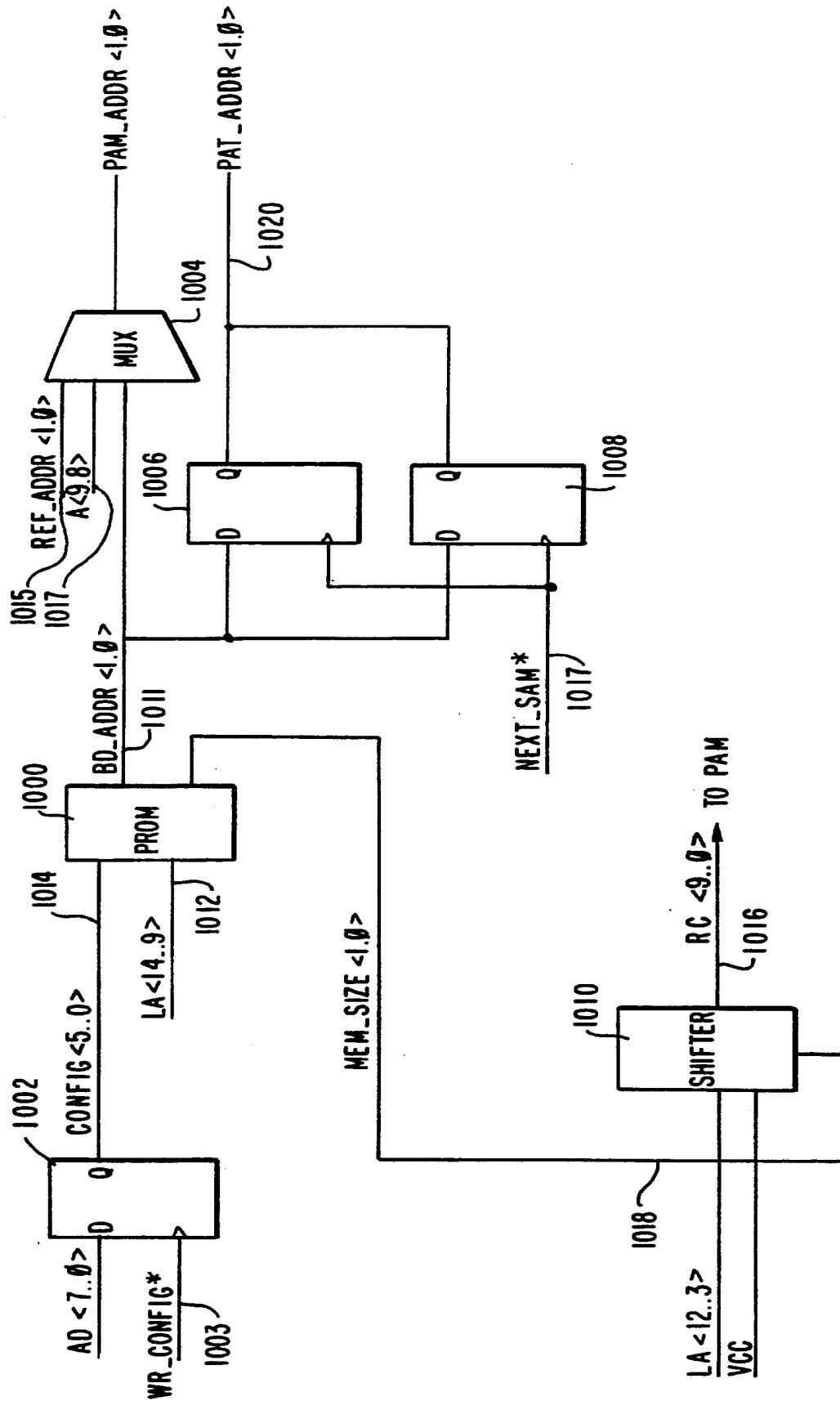
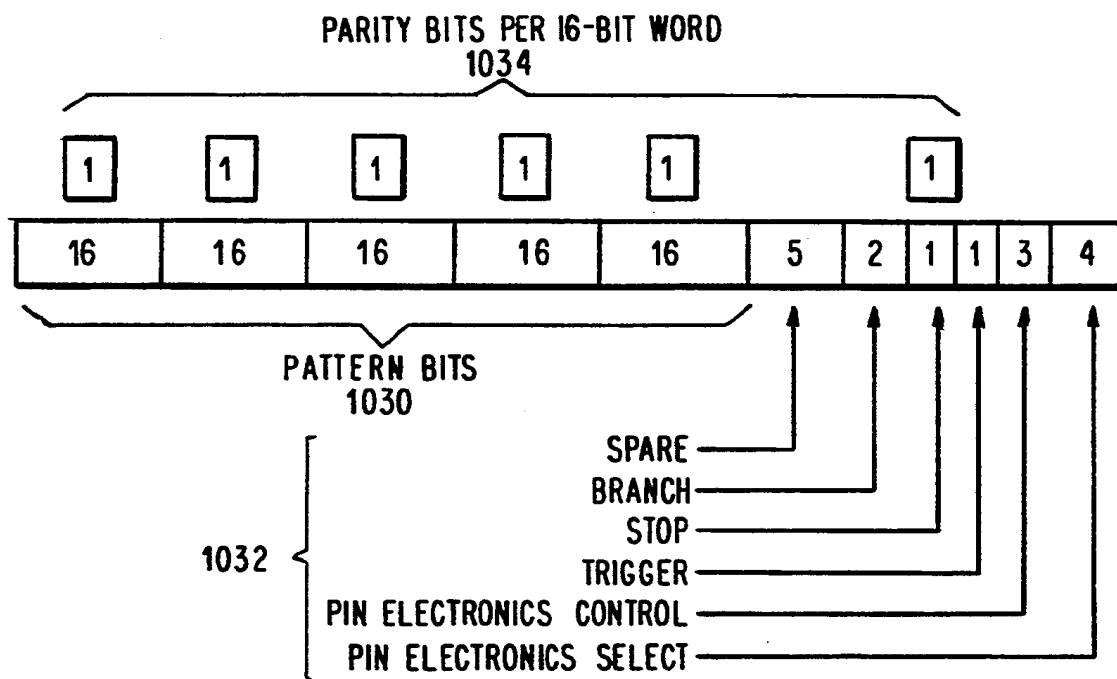


FIG. 16

**FIG.17**

1032 ↙

D15	BACKPLANE SPARE <1>	} 1048
D14	BACKPLANE SPARE <0>	
D13	PAC SPARE <2>	} 1046
D12	PAC SPARE <1>	
D11	PAC SPARE <0>	
D10	BRANCH <1>	} 1044
D9	BRANCH <0>	
D8	STOP	} 1042
D7	USER	
D6	PEL_CONTROL <2>	} 1038
D5	PEL_CONTROL <1>	
D4	PEL_CONTROL <0>	
D3	PEL_ADDRESS <3>	} 1036
D2	PEL_ADDRESS <2>	
D1	PEL_ADDRESS <1>	
D0	PEL_ADDRESS <0>	

FIG.18

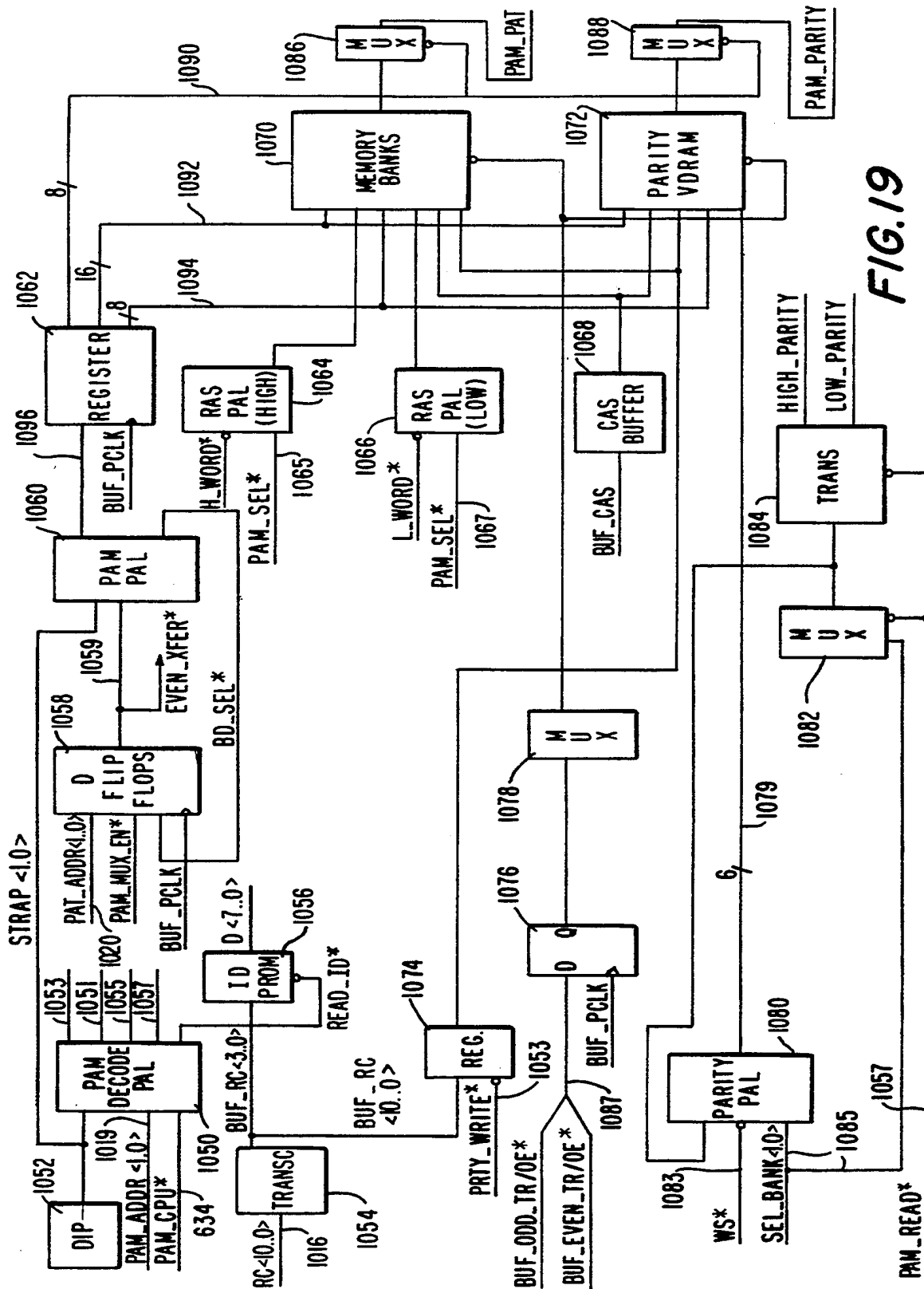


FIG. 19

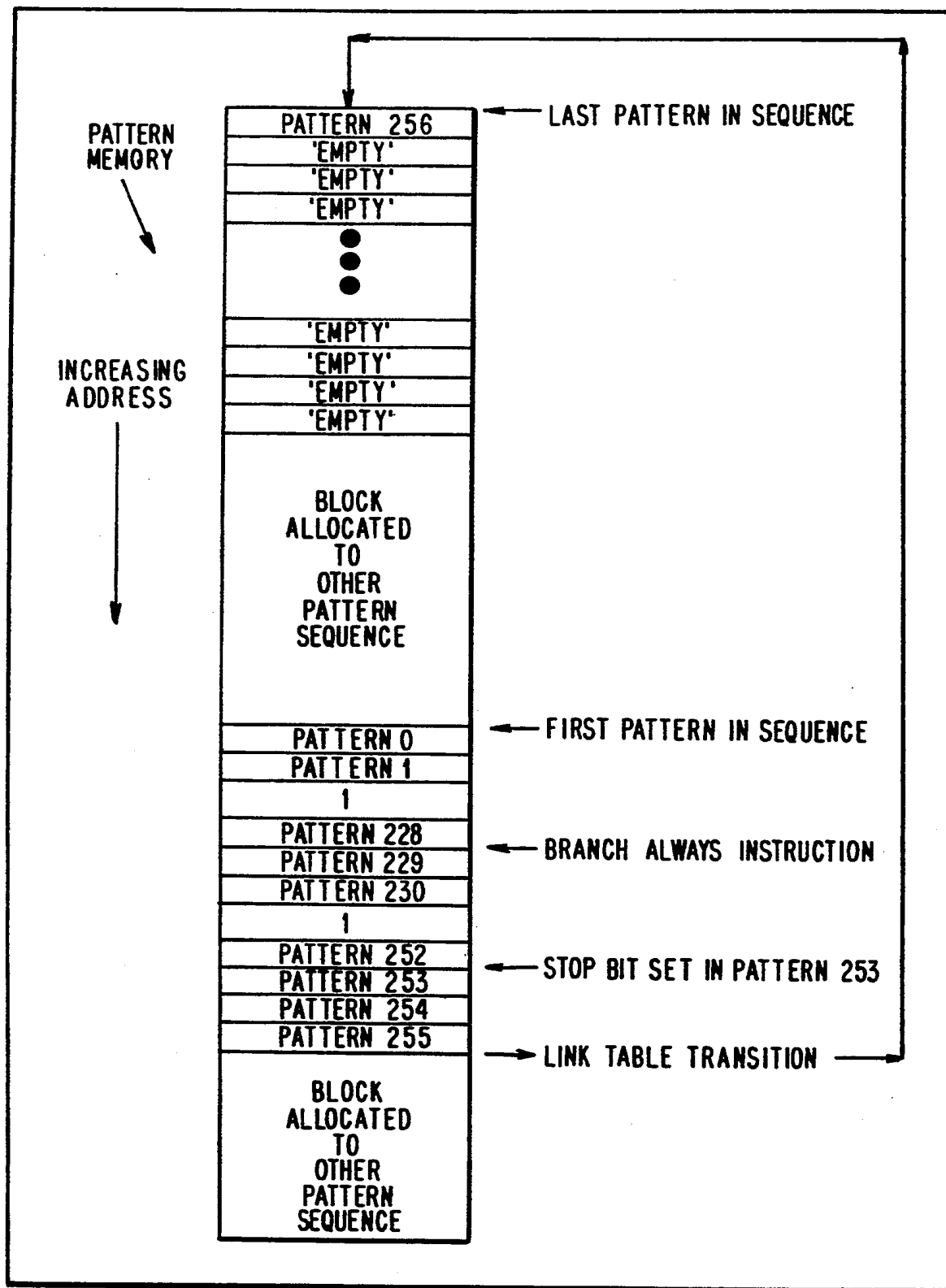
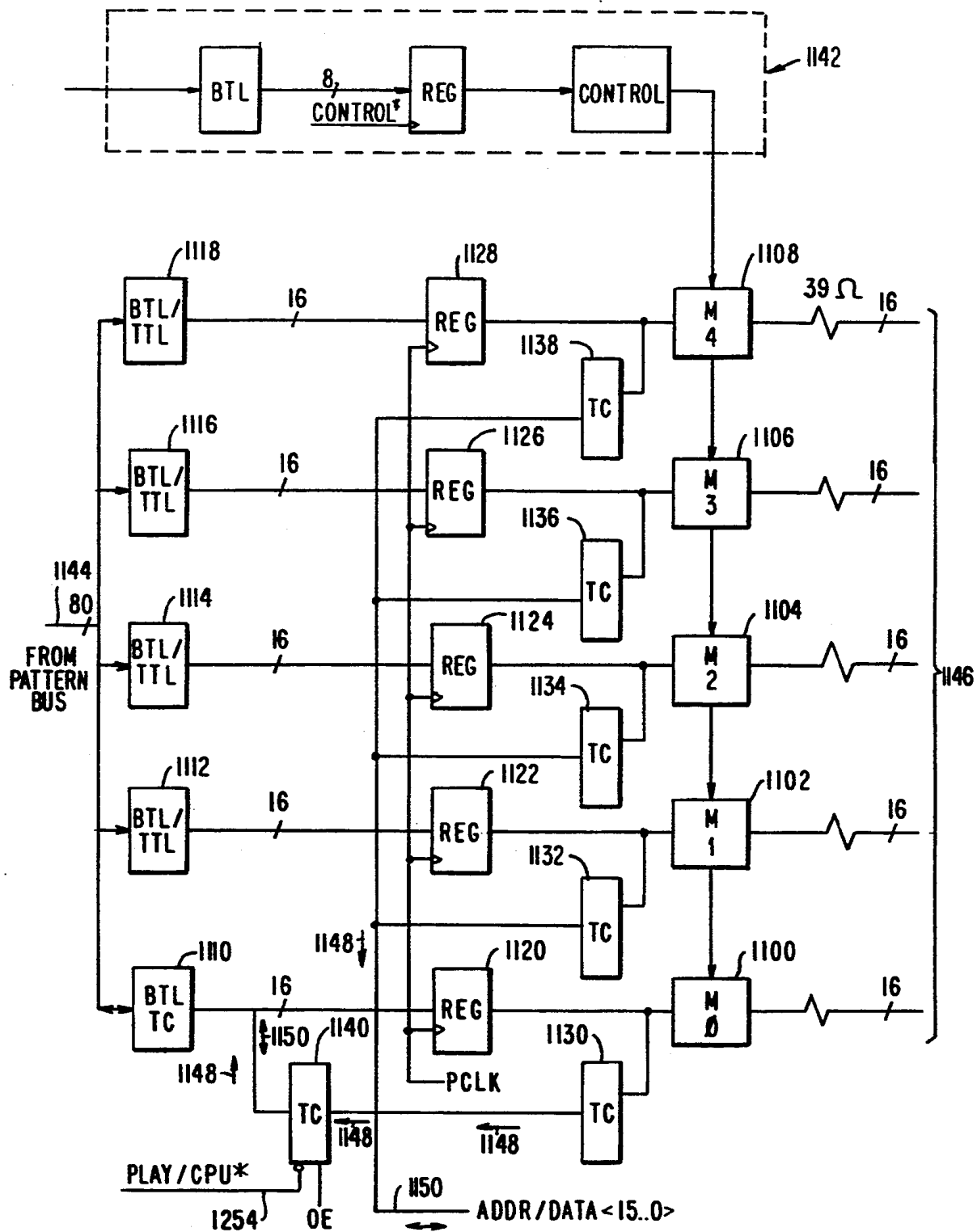
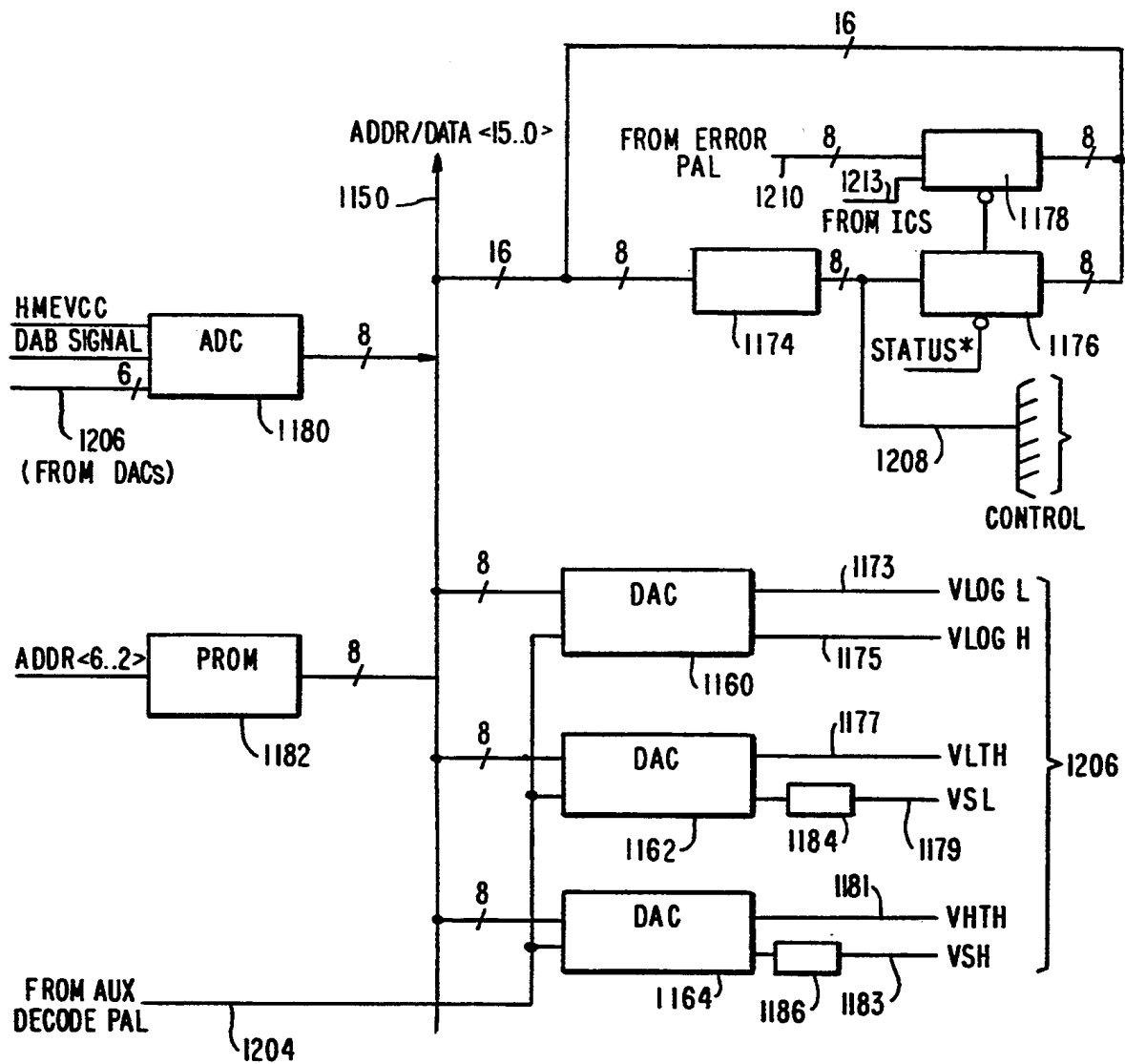


FIG. 20



**FIG. 22**

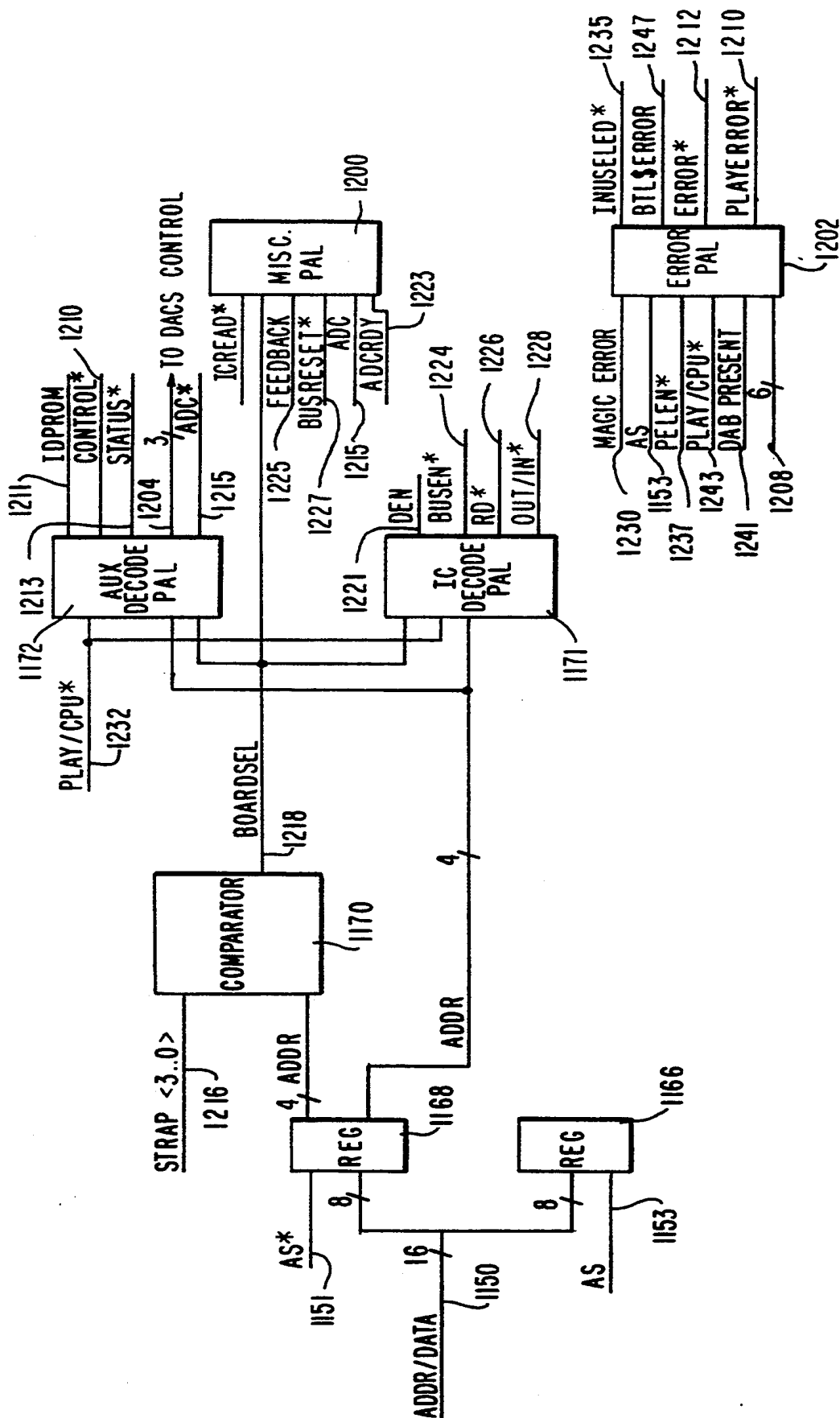


FIG. 23

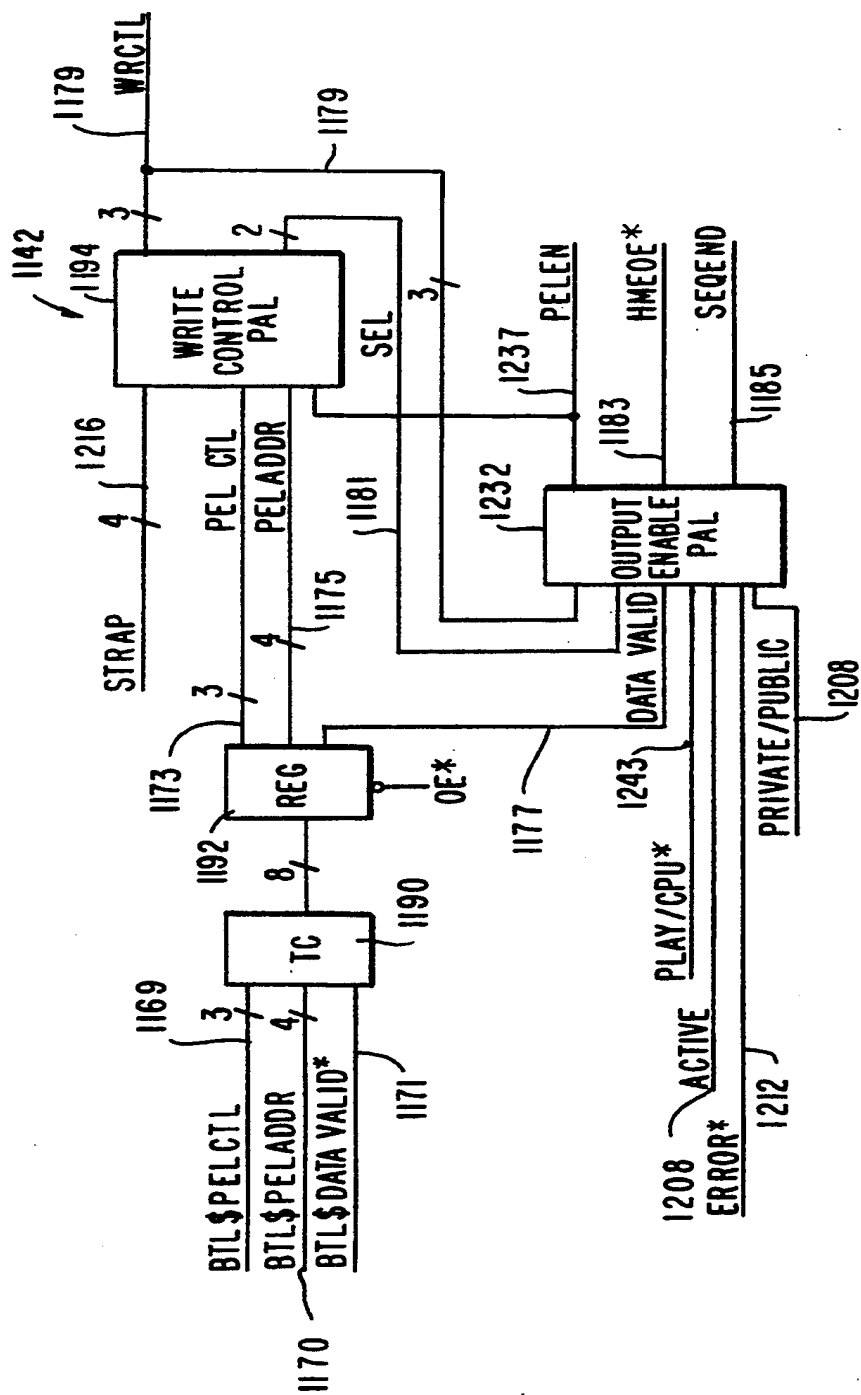
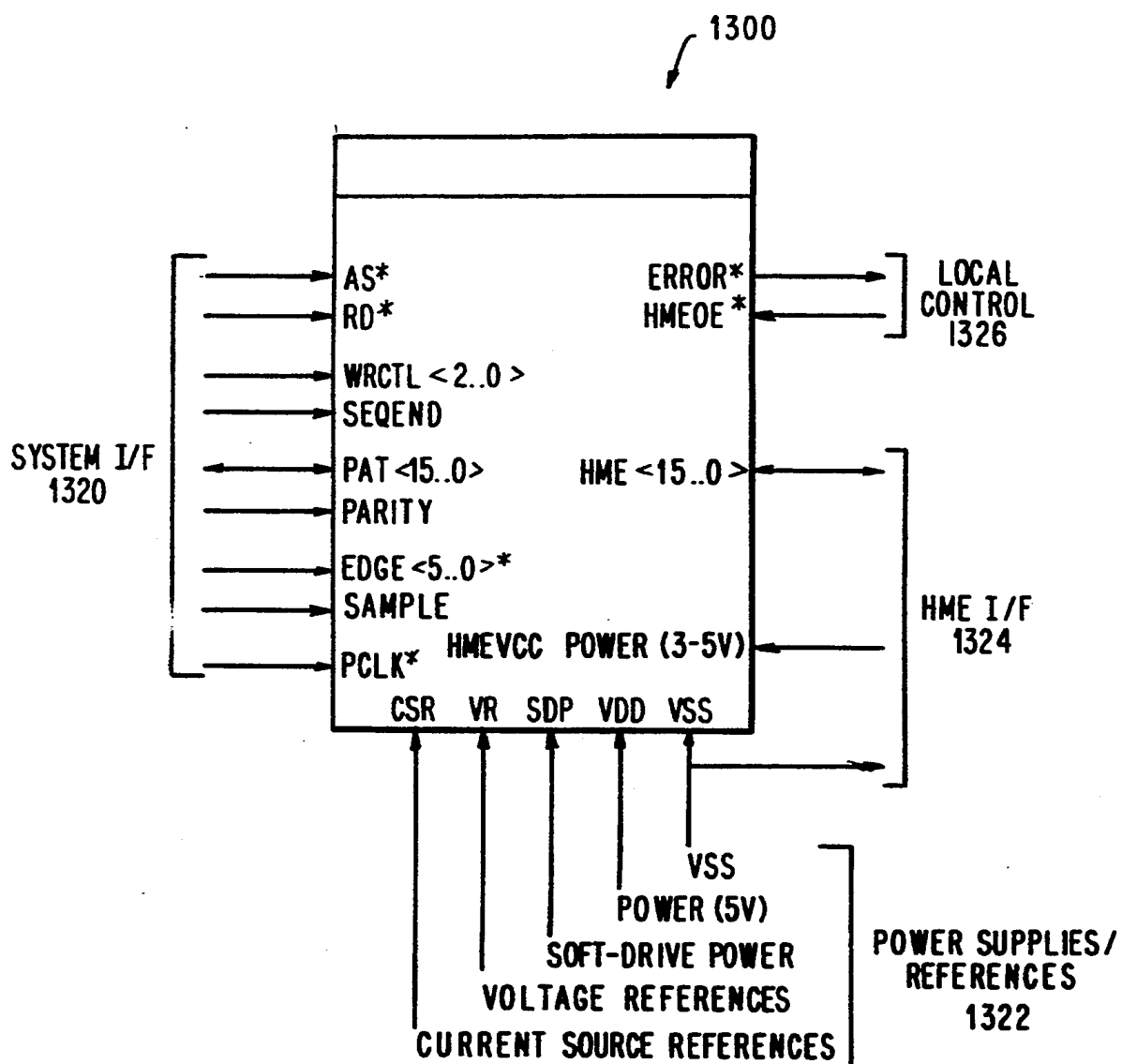
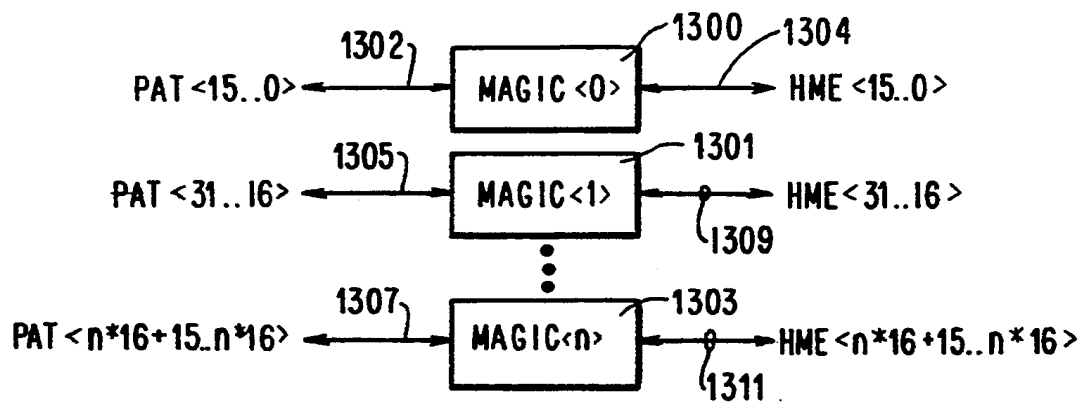
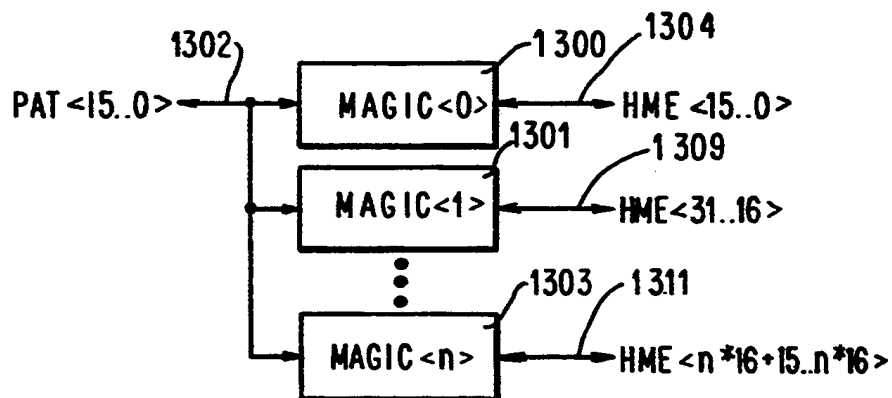


FIG. 24

**FIG.25**

**FIG. 26****FIG. 27**

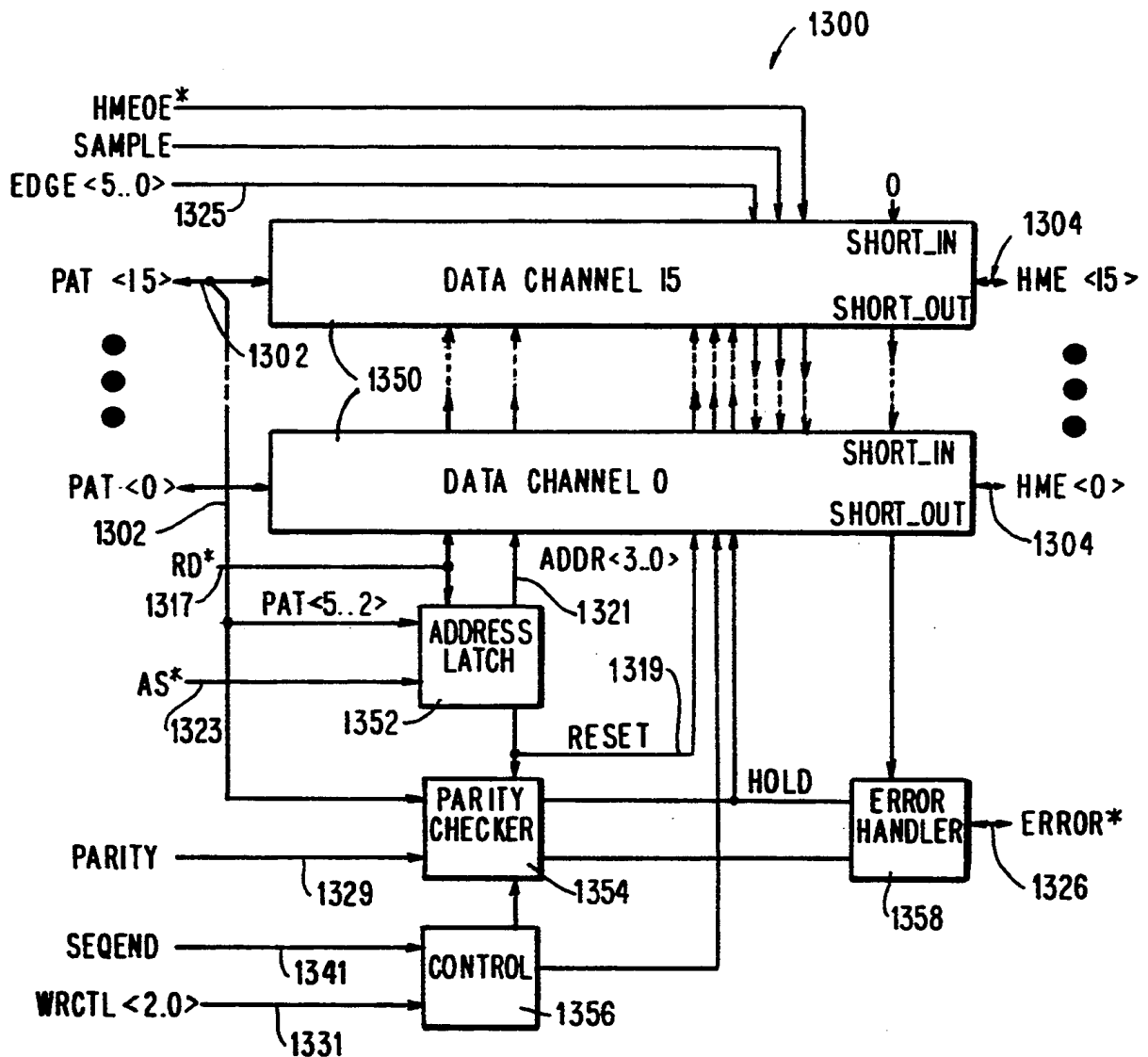


FIG. 28

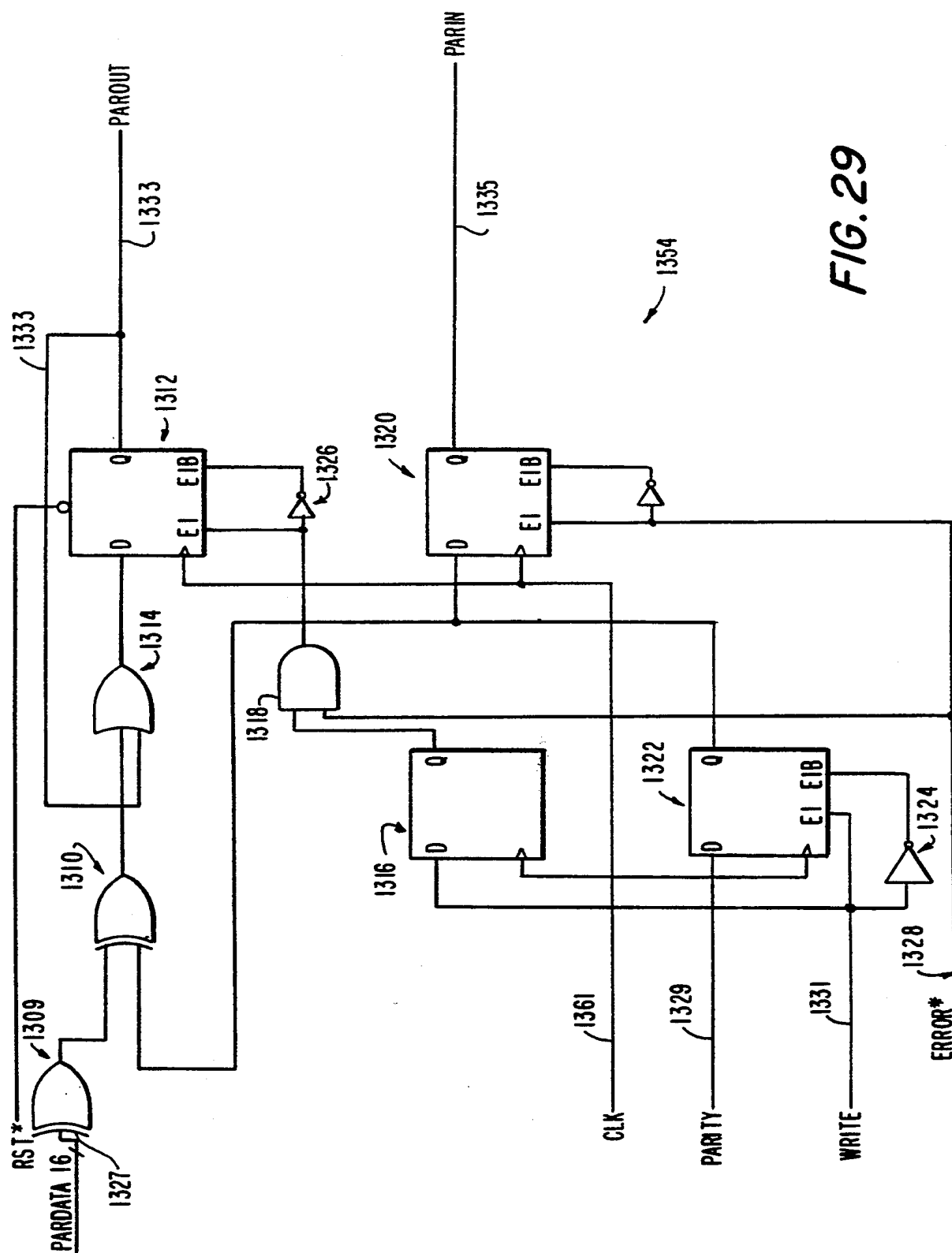
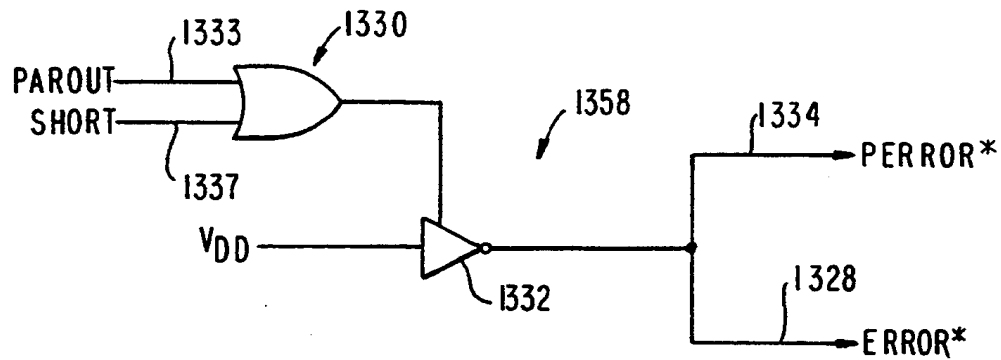
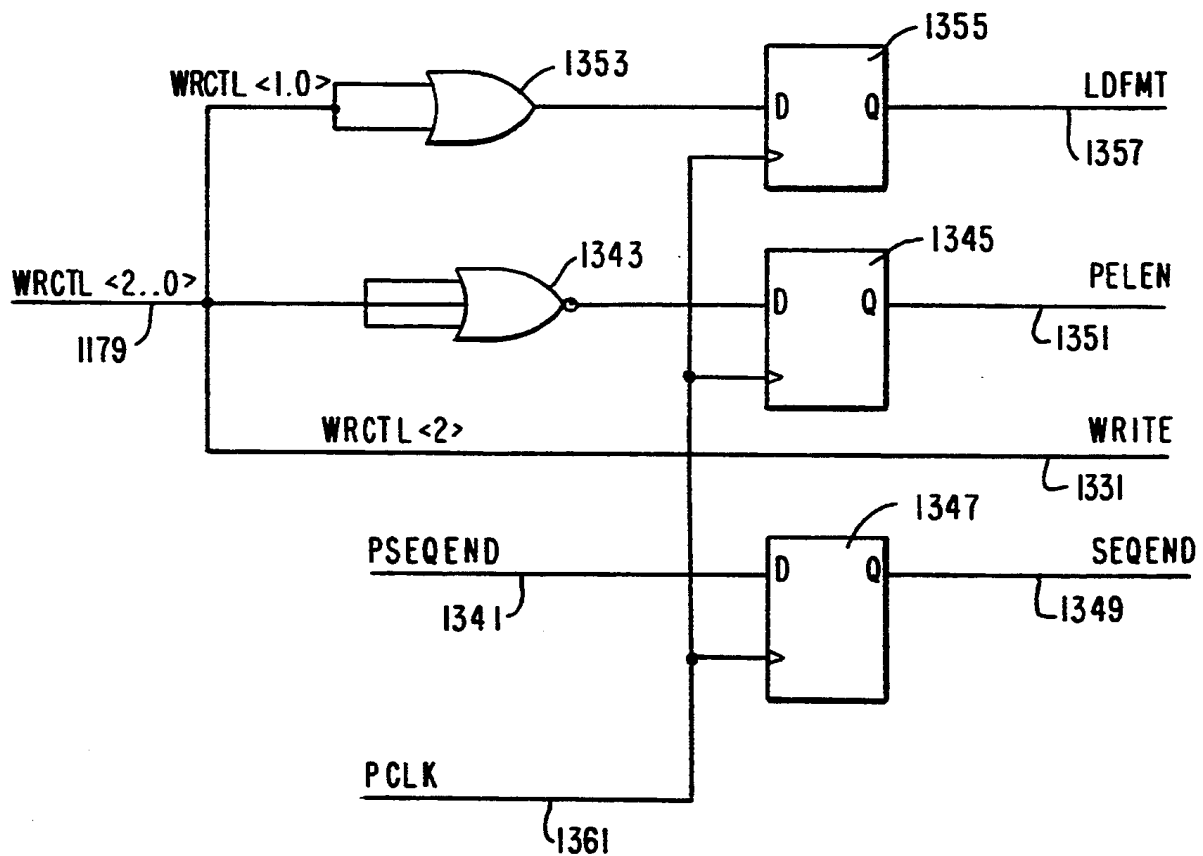


FIG. 29

**FIG. 30****FIG. 31**

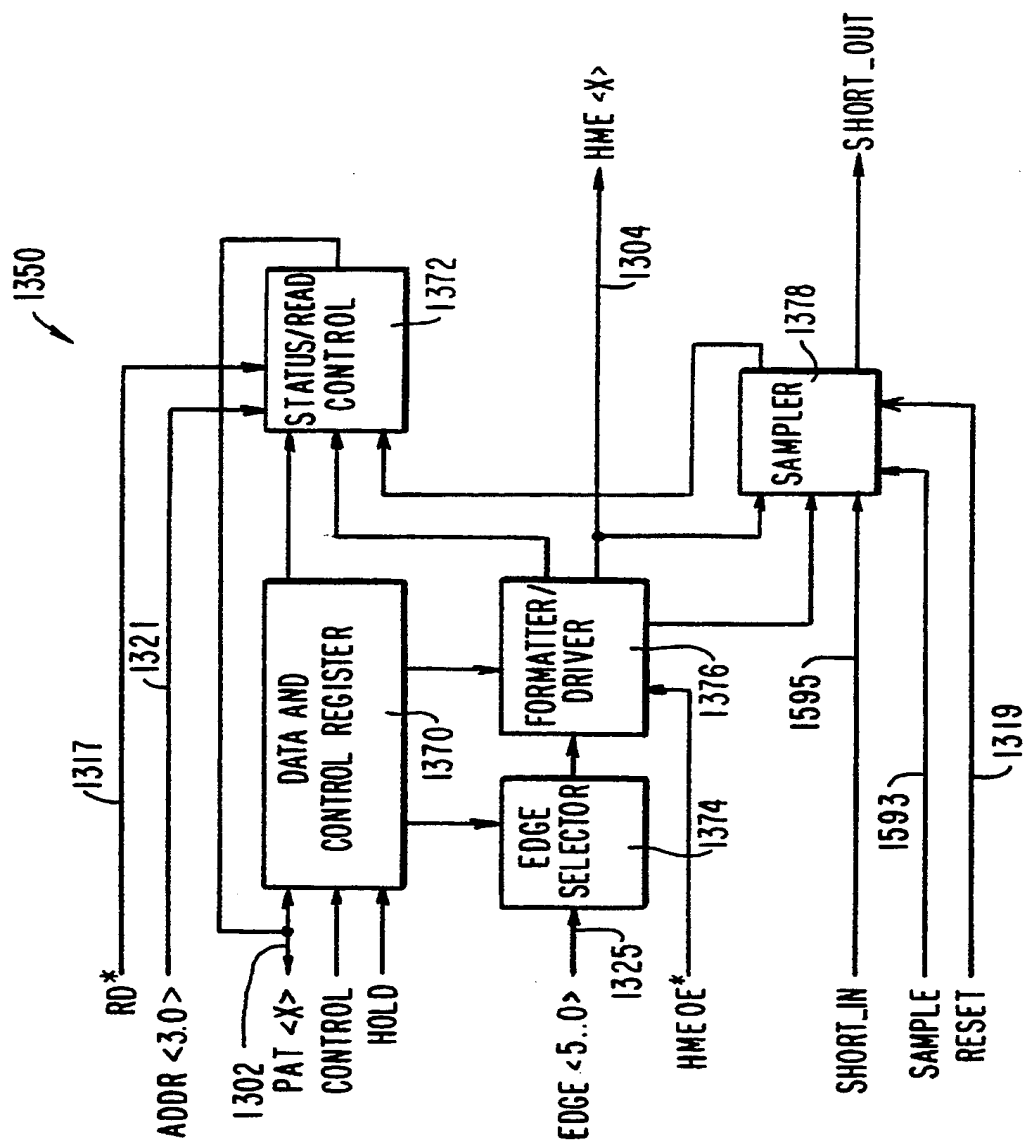


FIG. 32

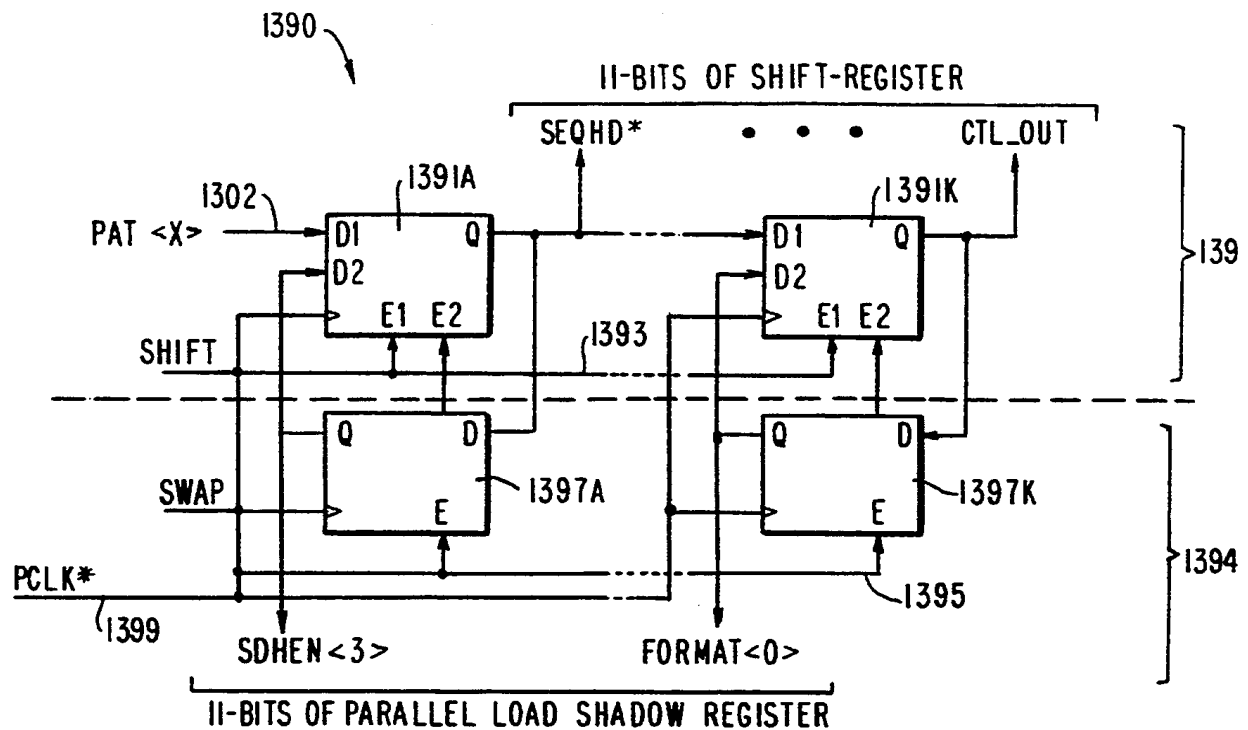


FIG. 33

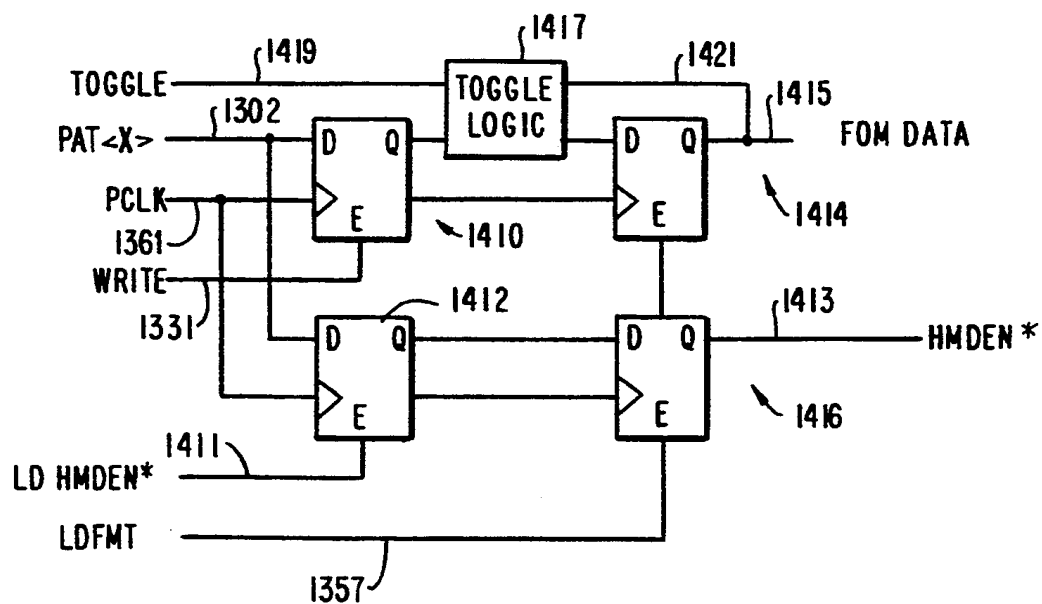
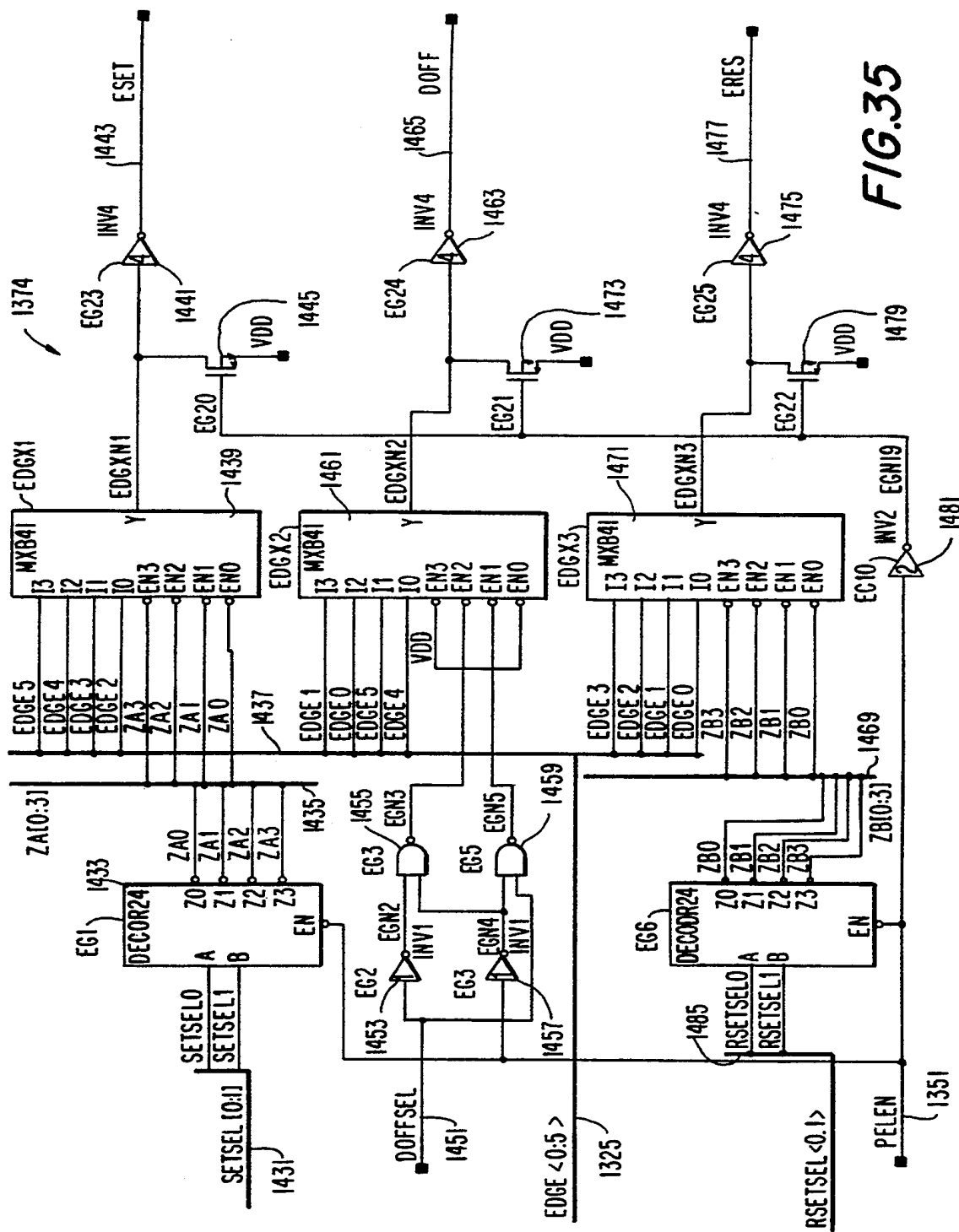


FIG. 34



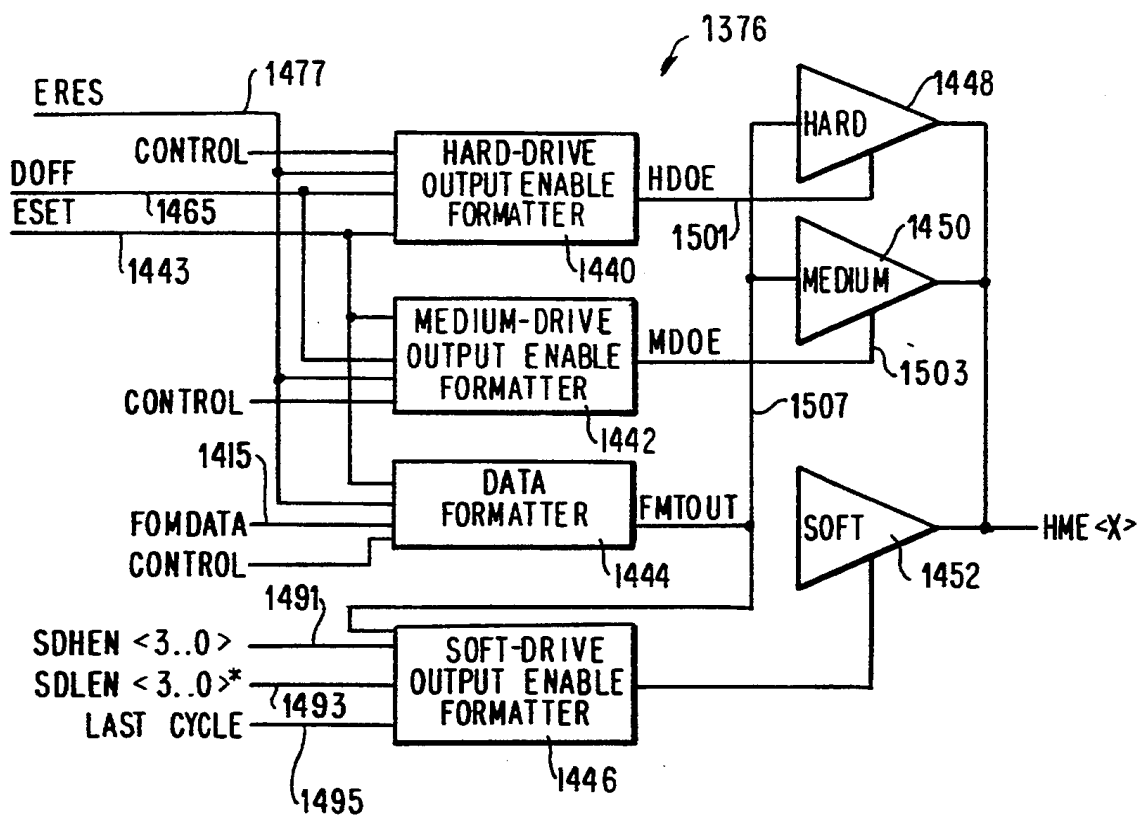


FIG. 36

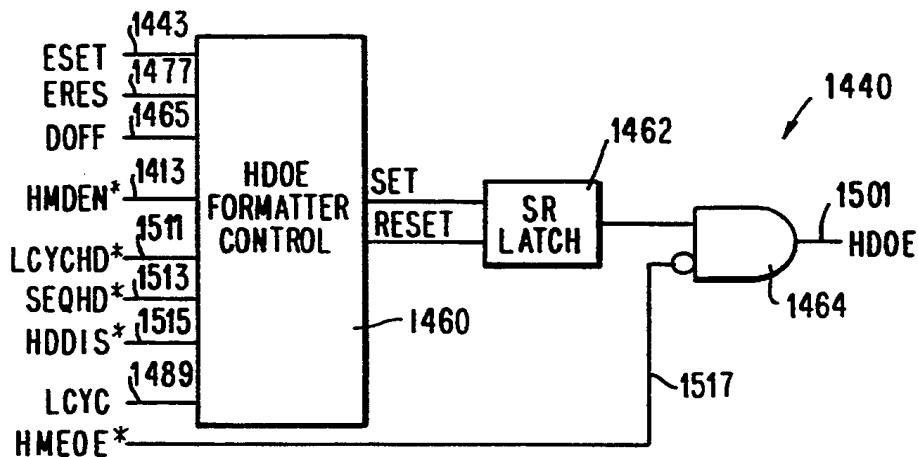
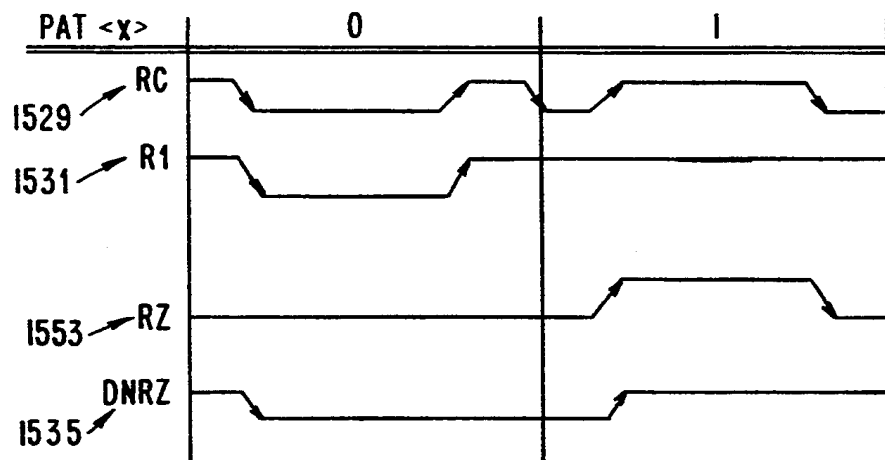
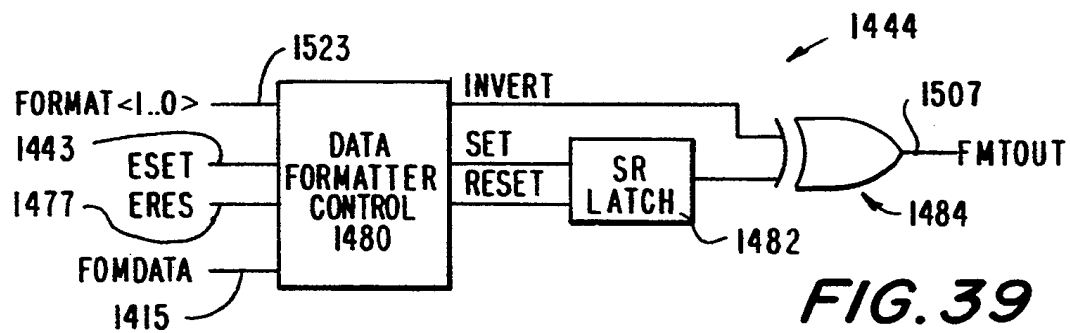
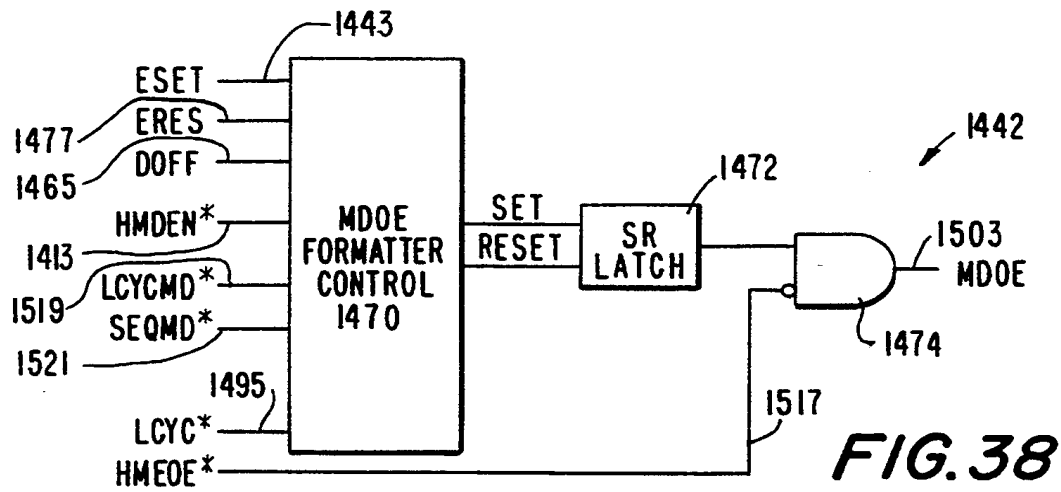


FIG. 37



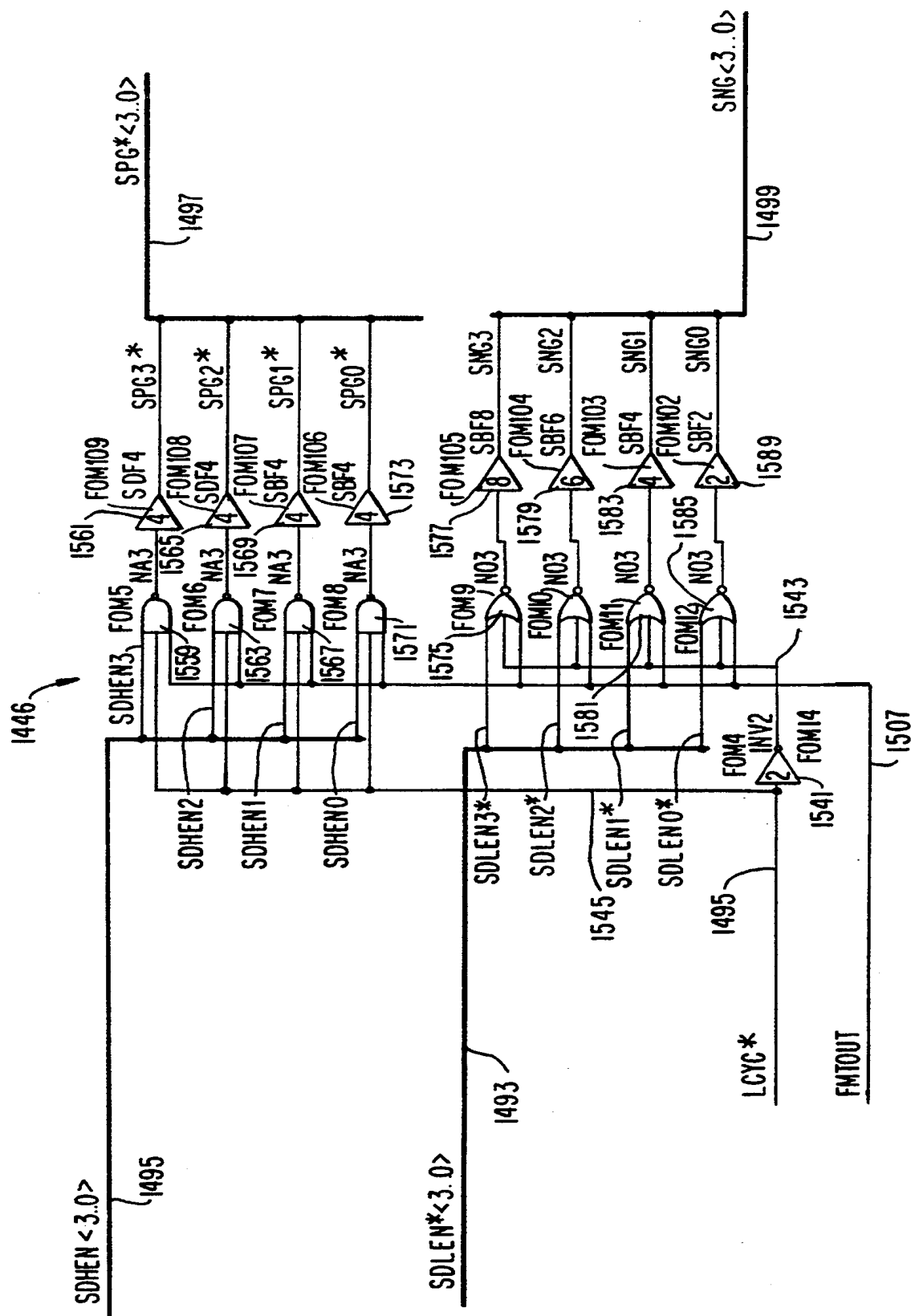
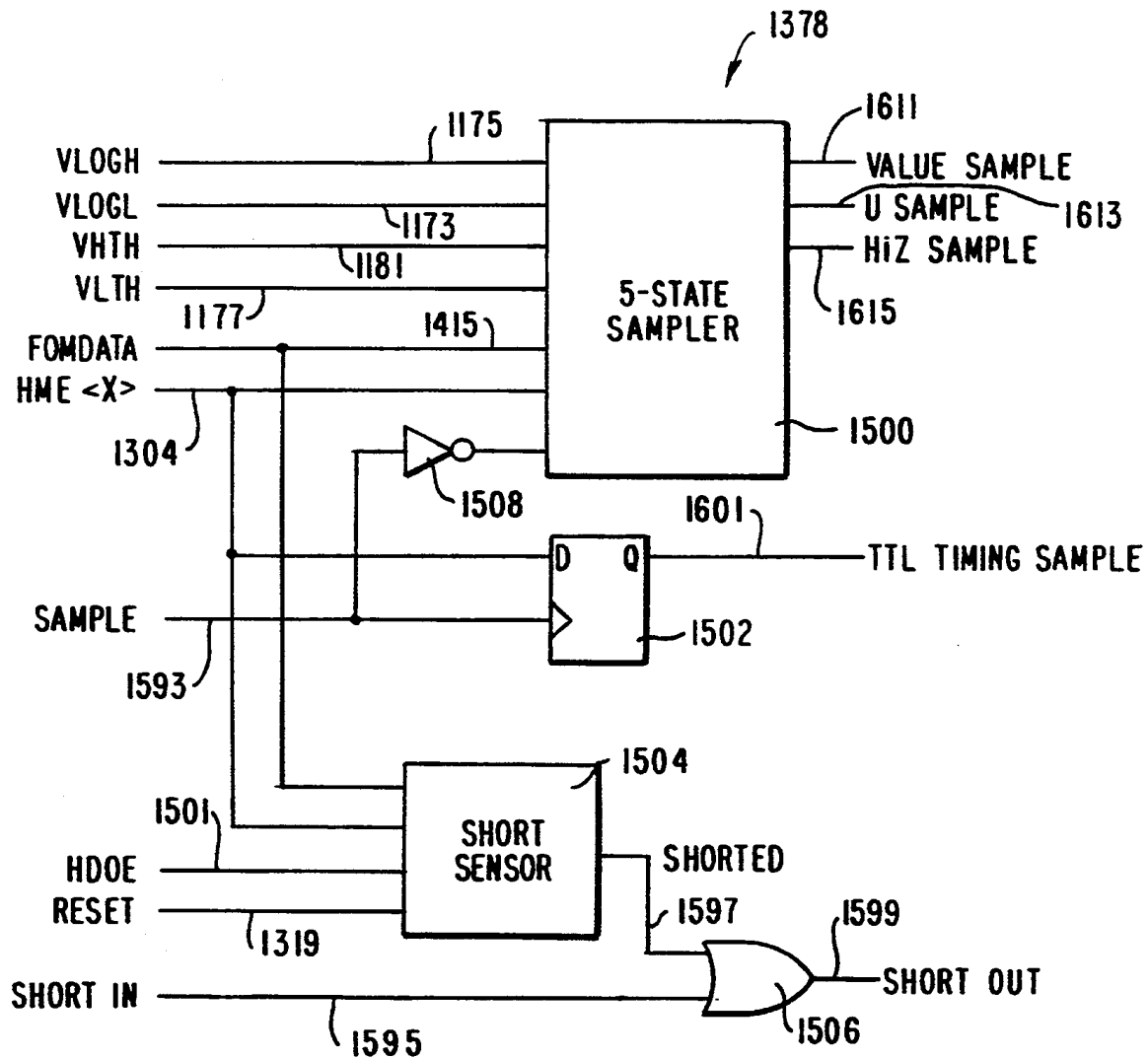
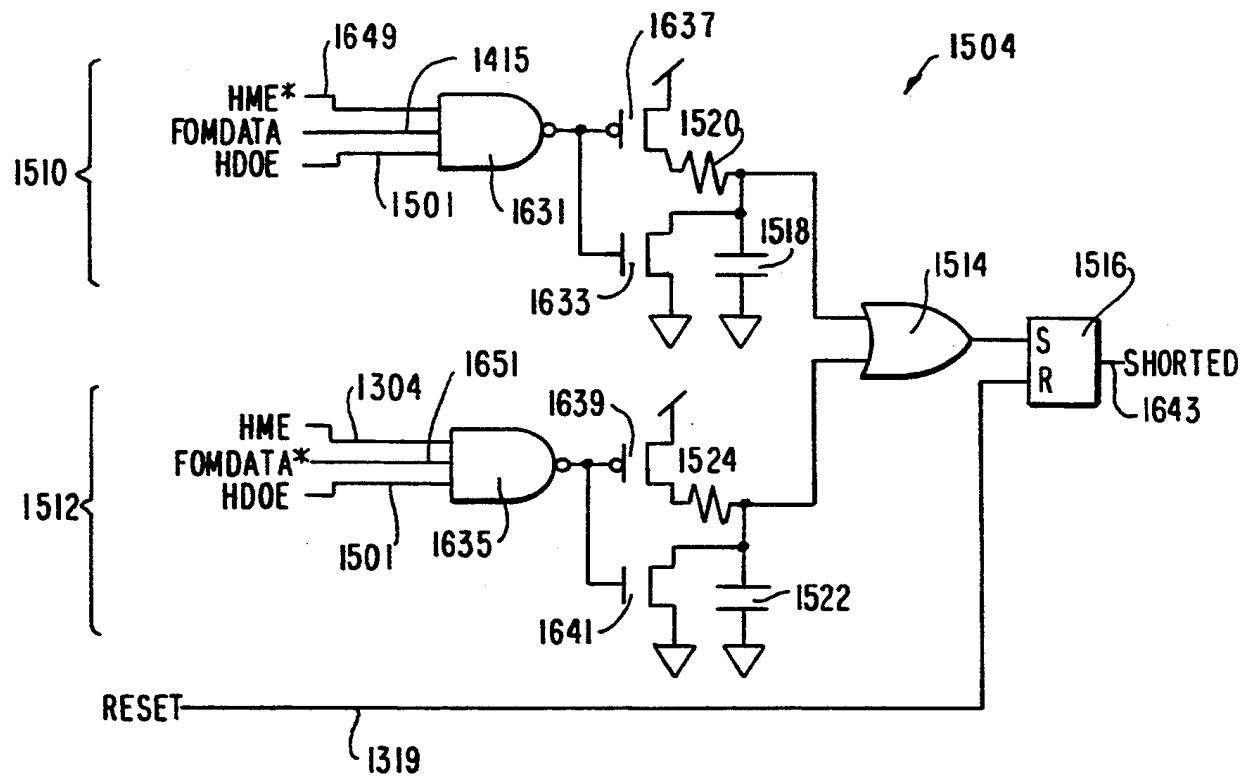
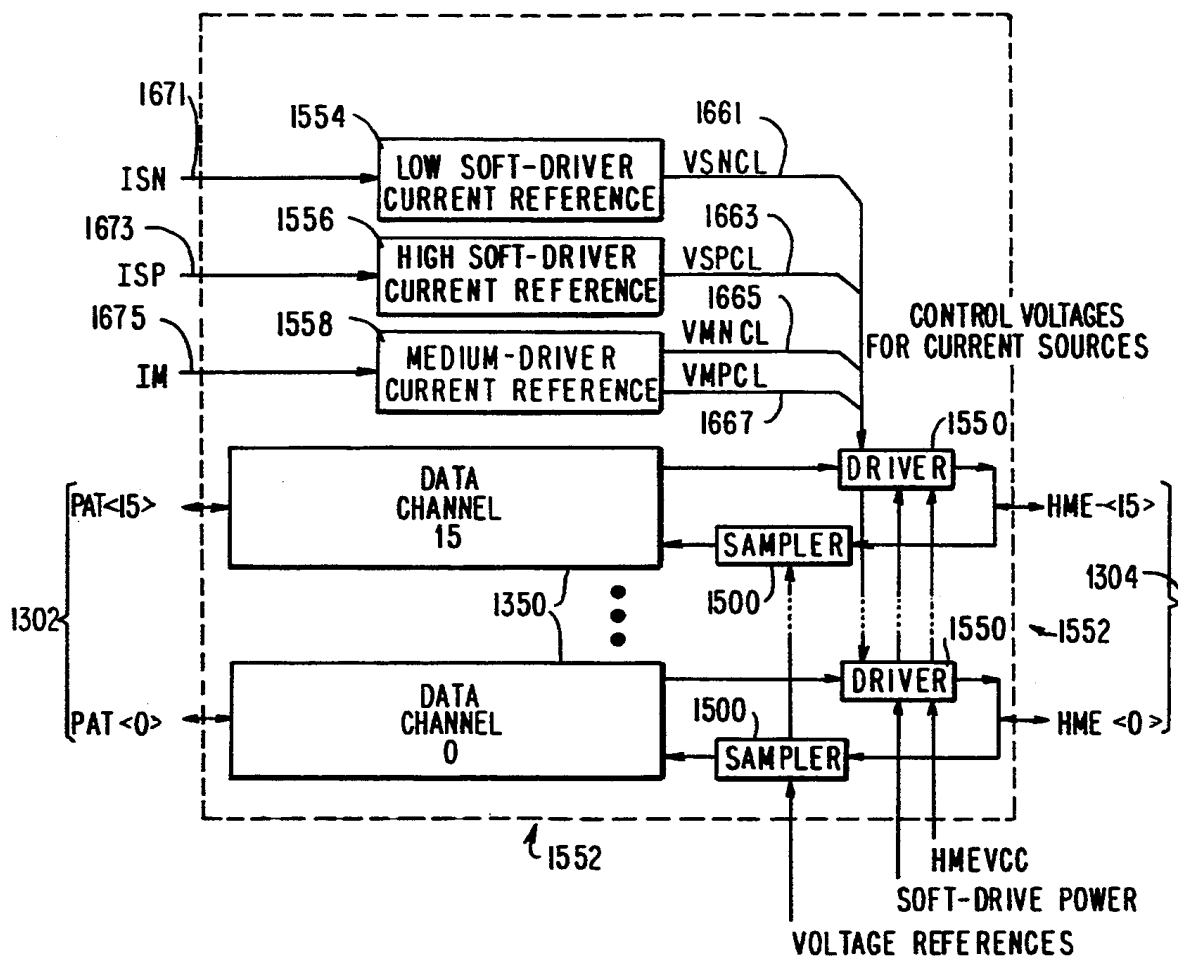
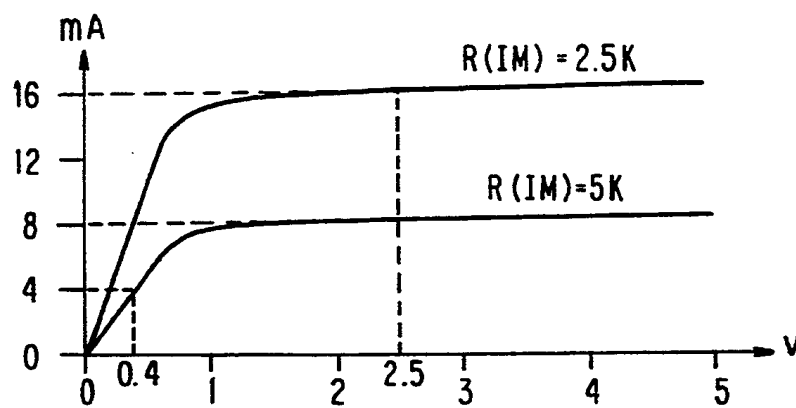
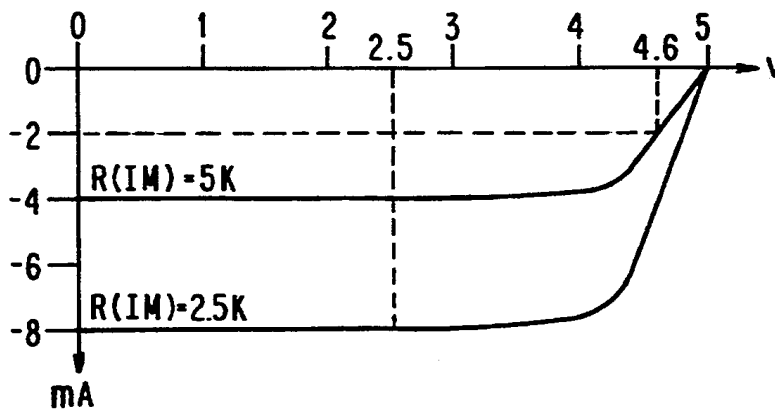


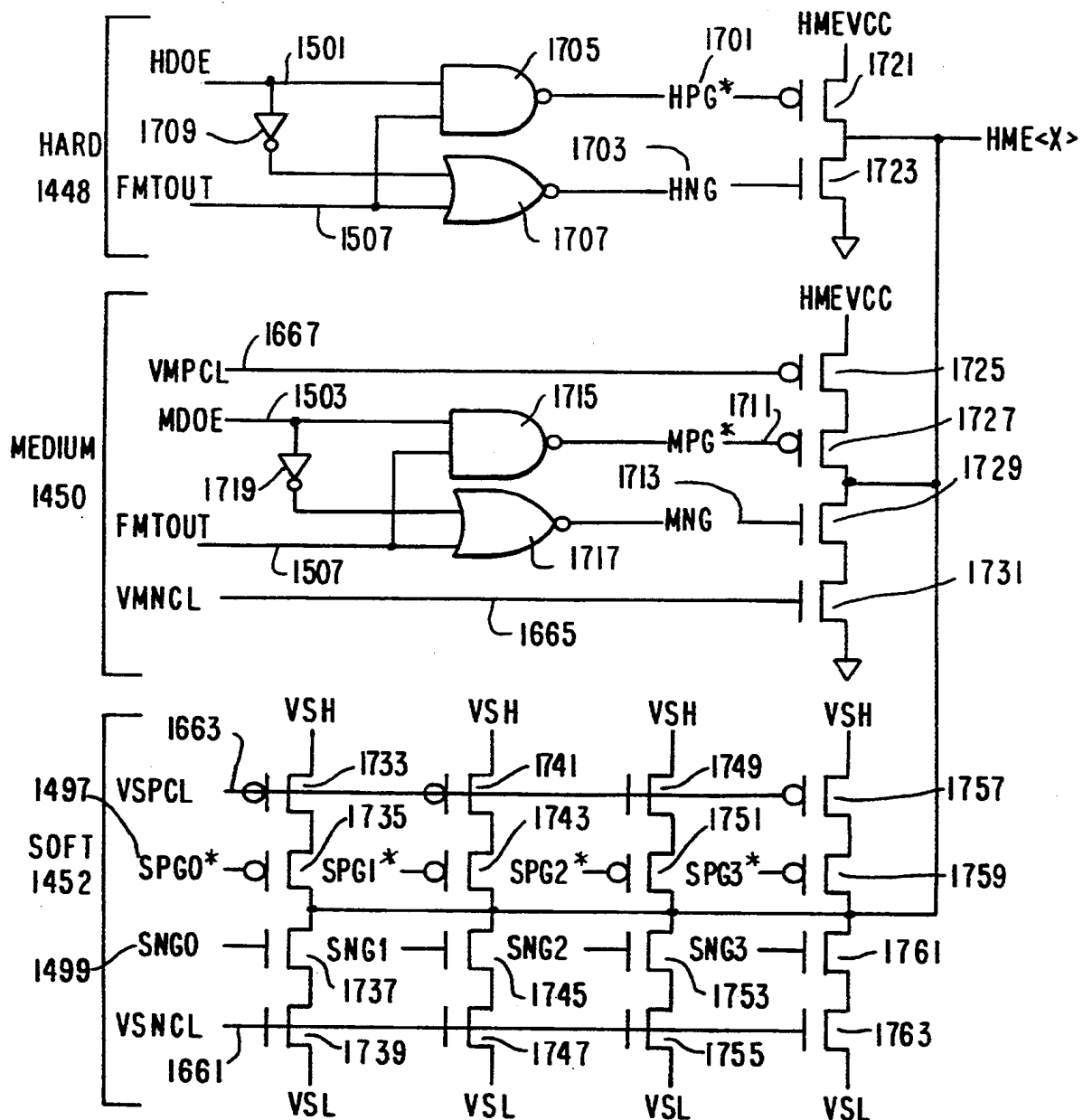
FIG. 41

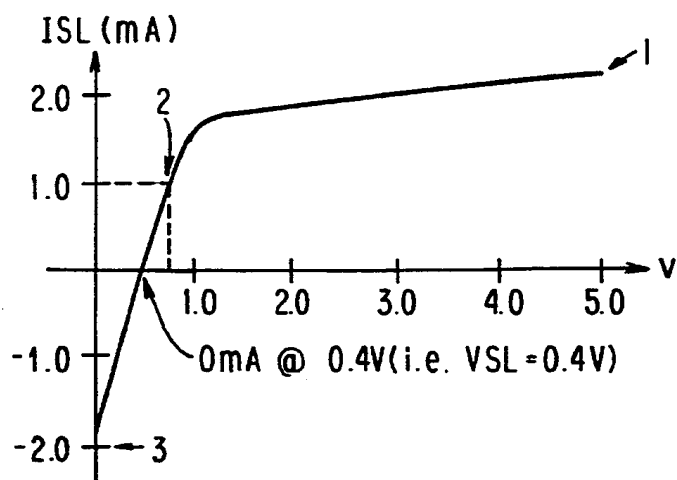
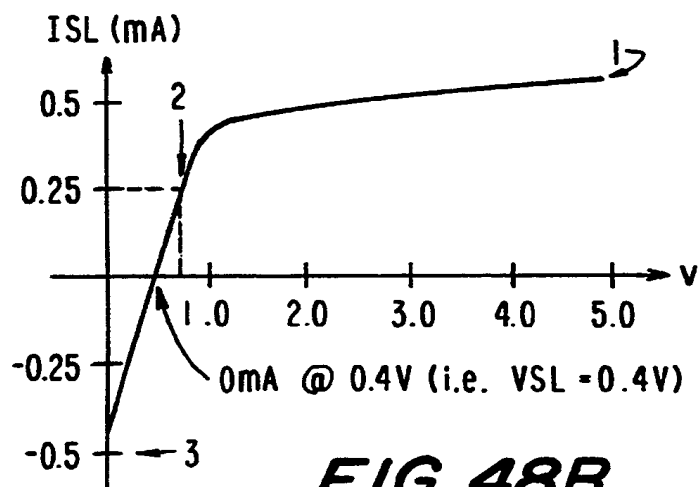
**FIG. 42**

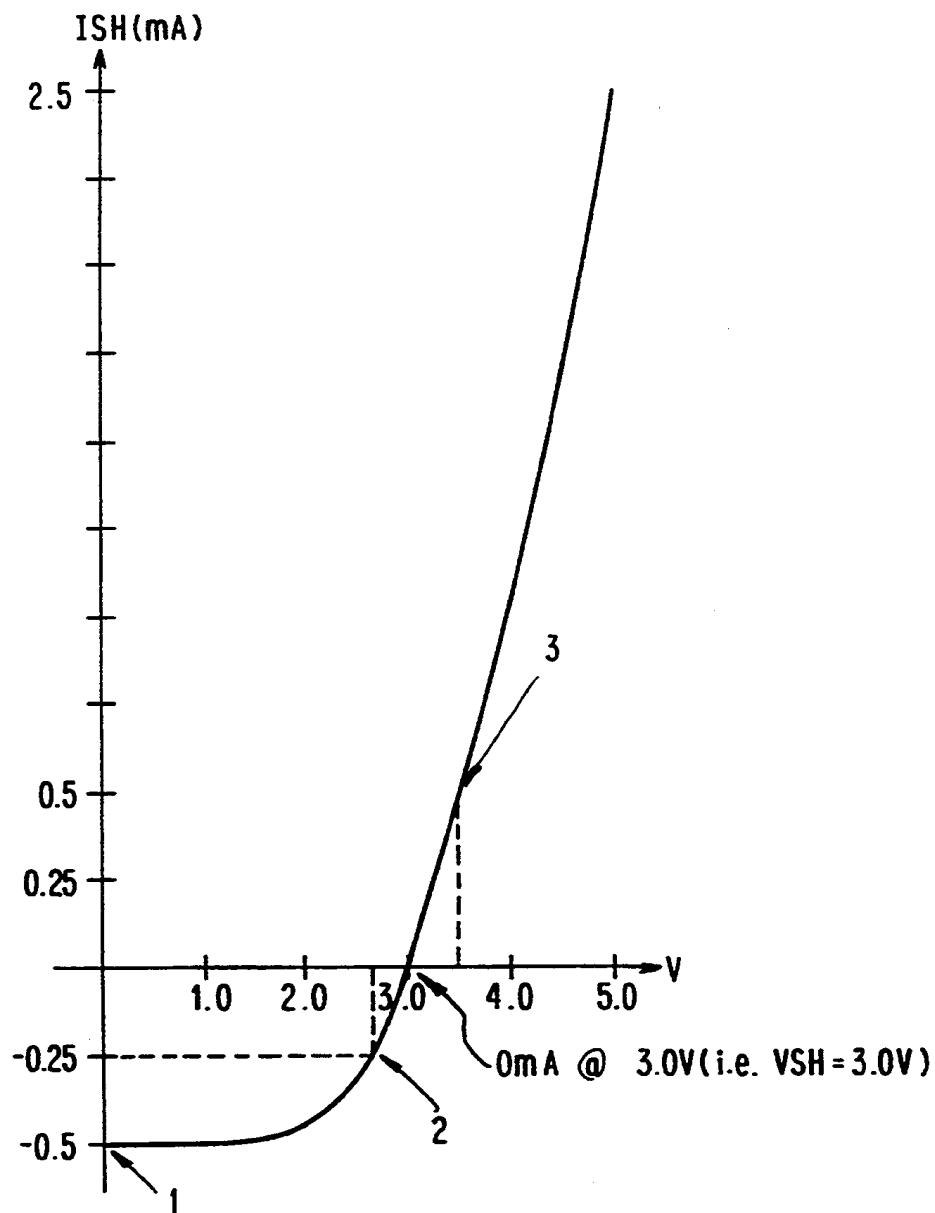
**FIG. 43**

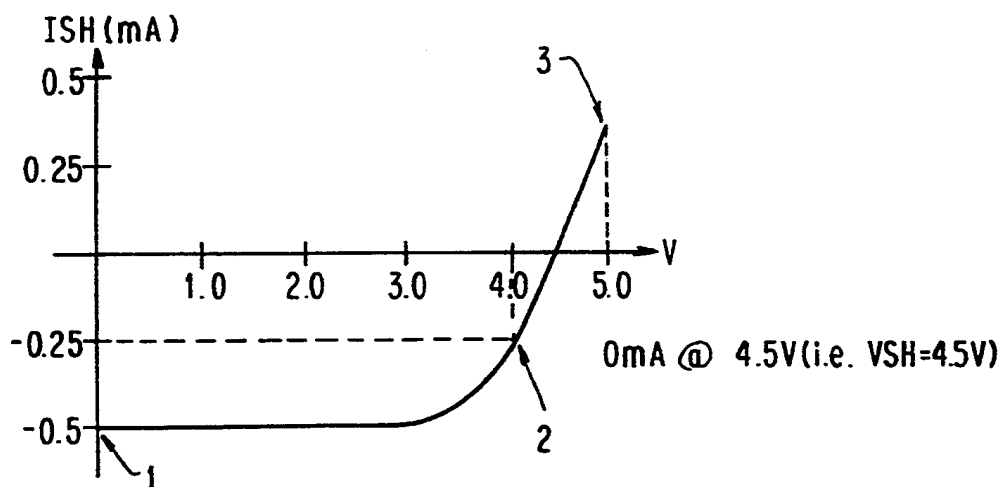
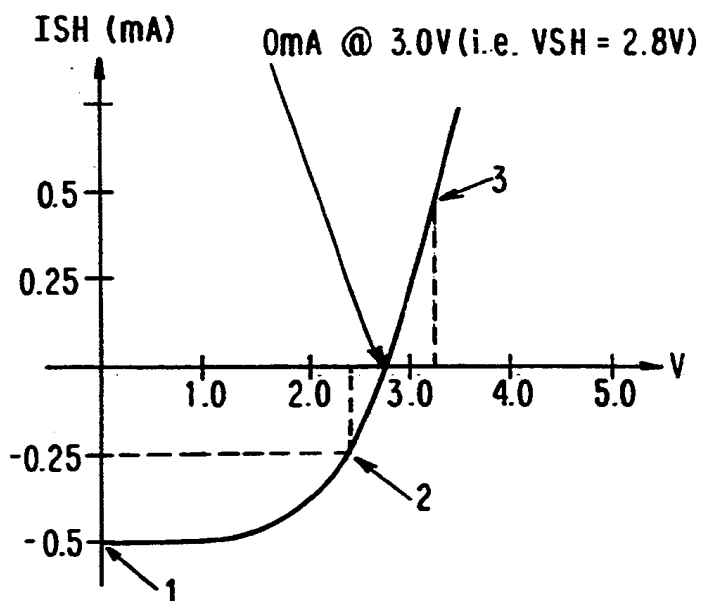
**FIG. 44**

**FIG. 45****FIG. 46**

**FIG. 47**

**FIG. 48A****FIG. 48B**

**FIG.48C**

**FIG. 48D****FIG. 48E**

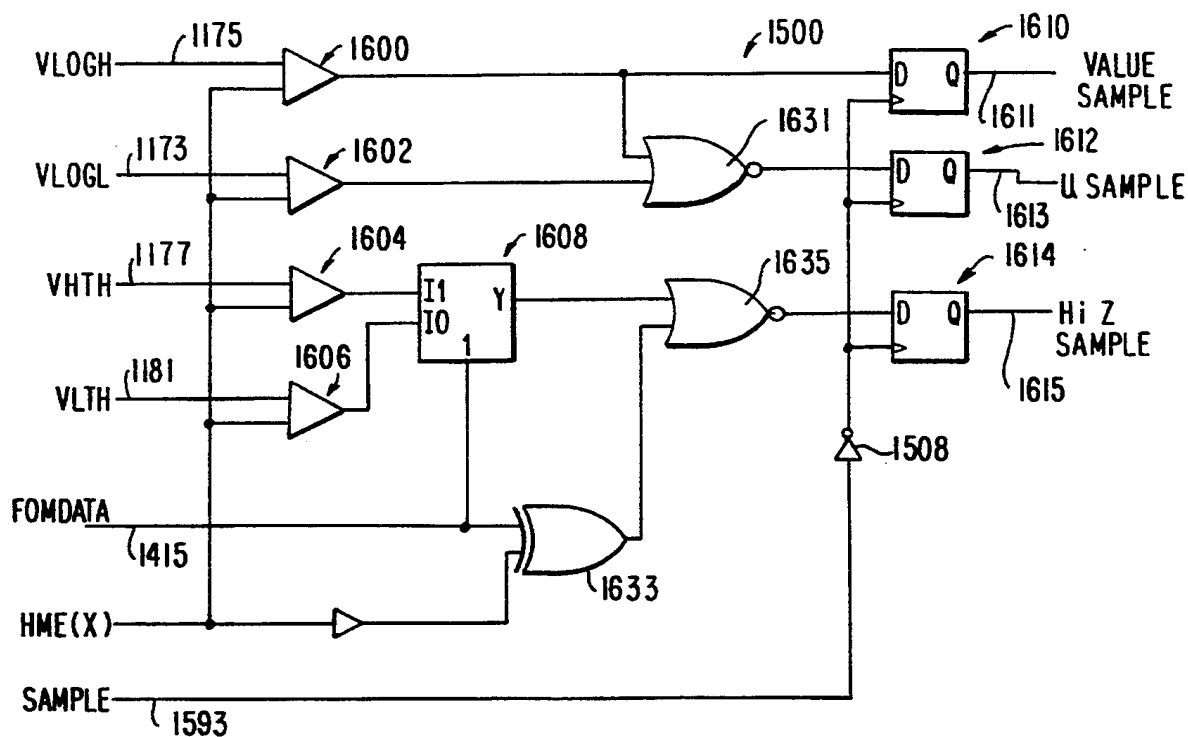


FIG. 49

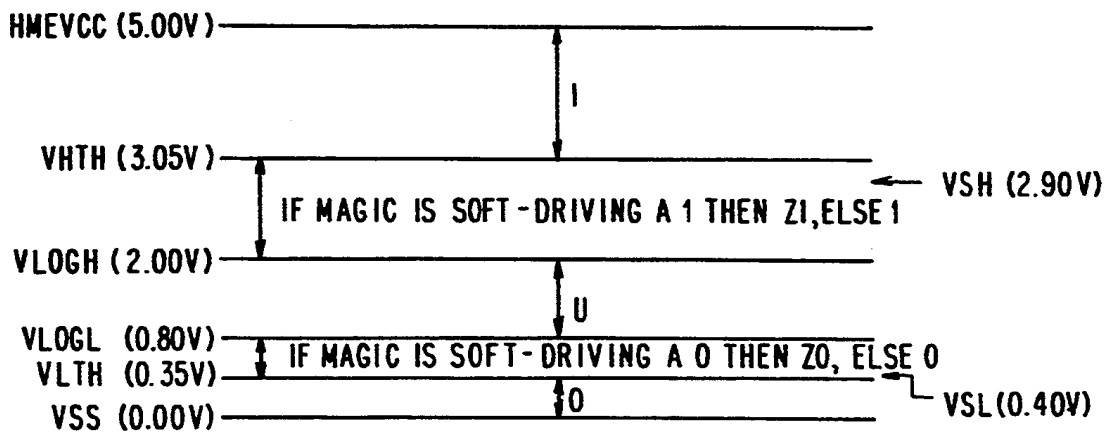
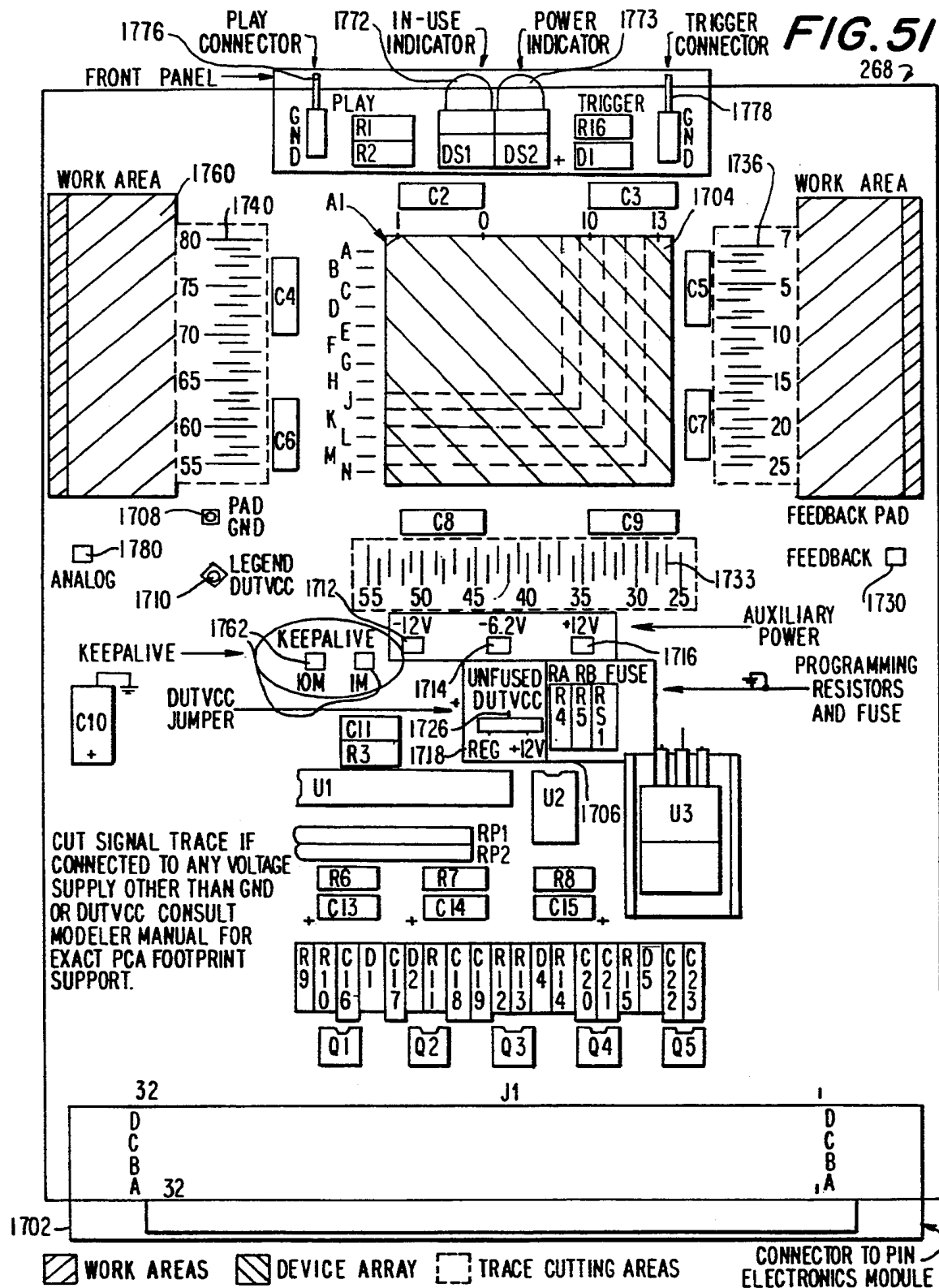
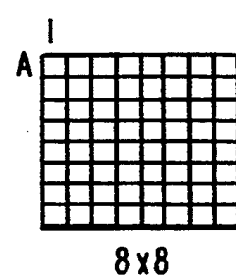
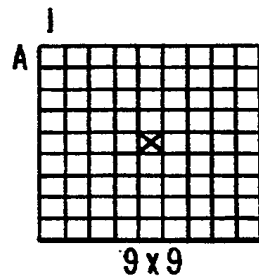
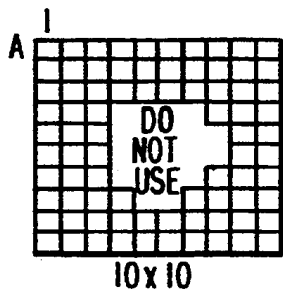
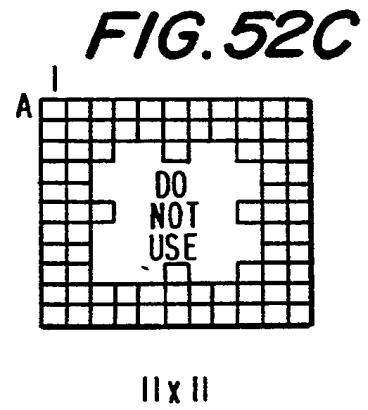
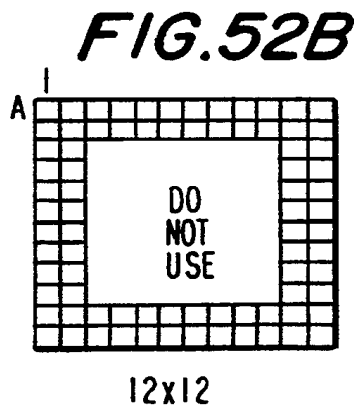
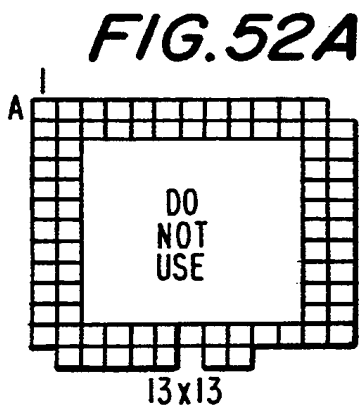
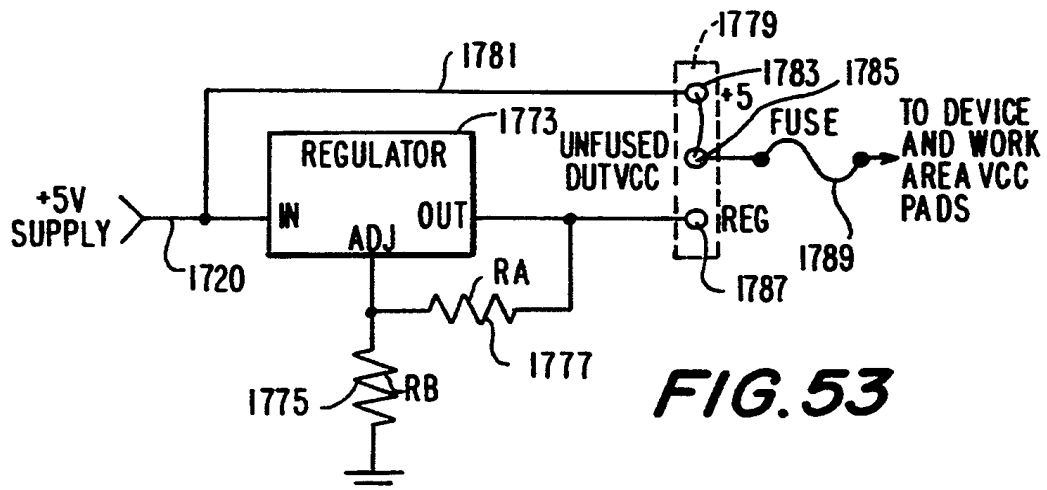


FIG. 50



**FIG. 52D****FIG. 52E****FIG. 52F**

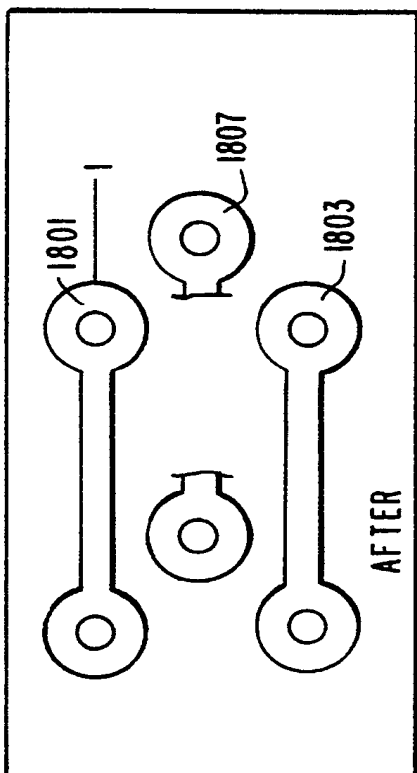


FIG. 54B

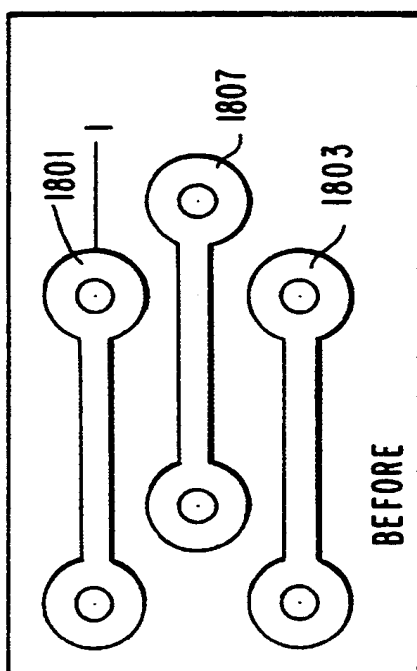


FIG. 54A

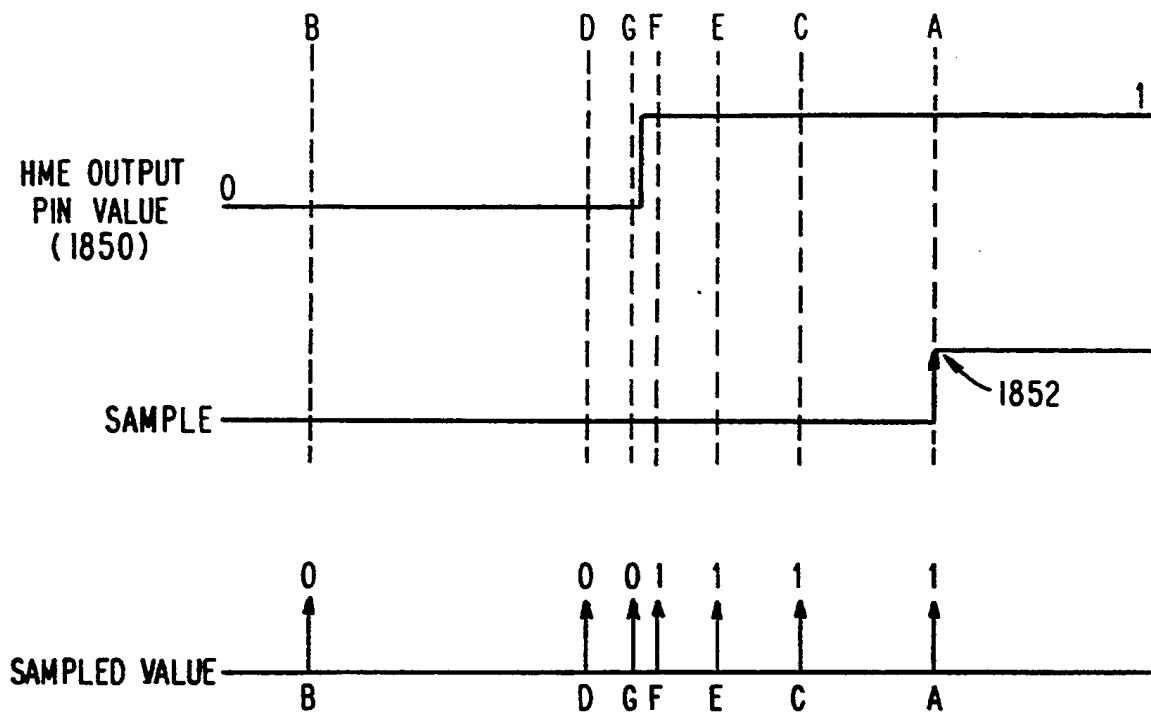


FIG. 55

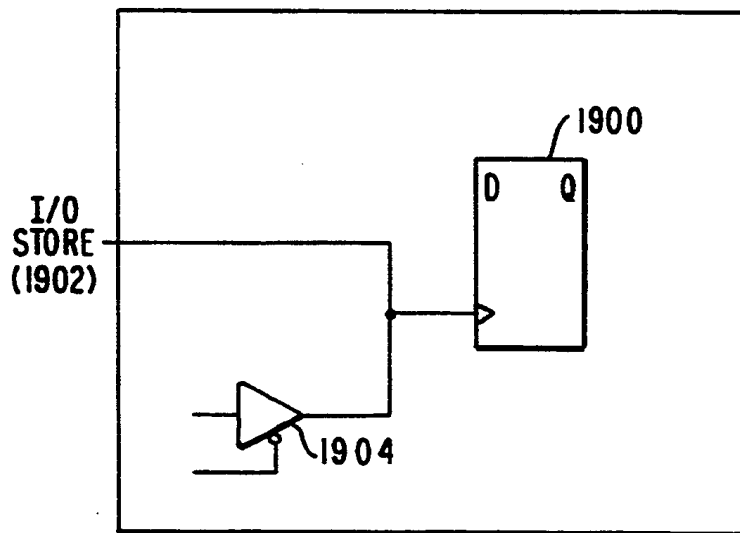


FIG. 56

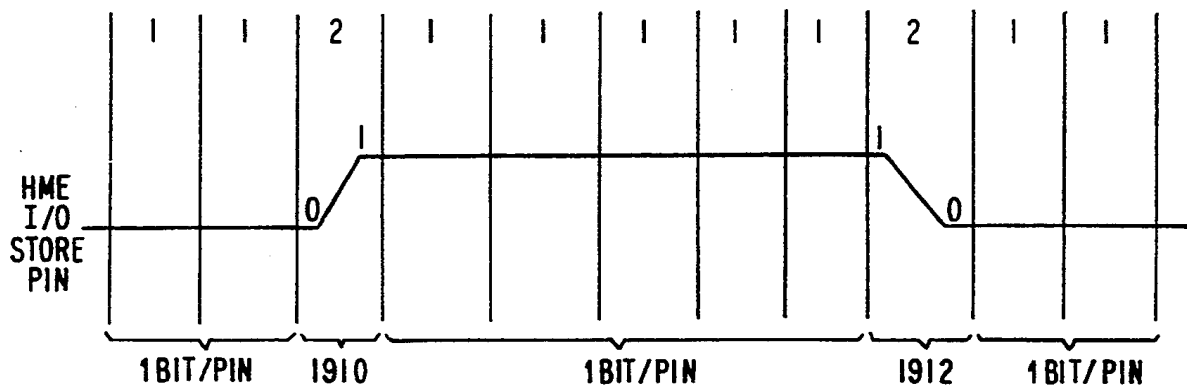


FIG. 57

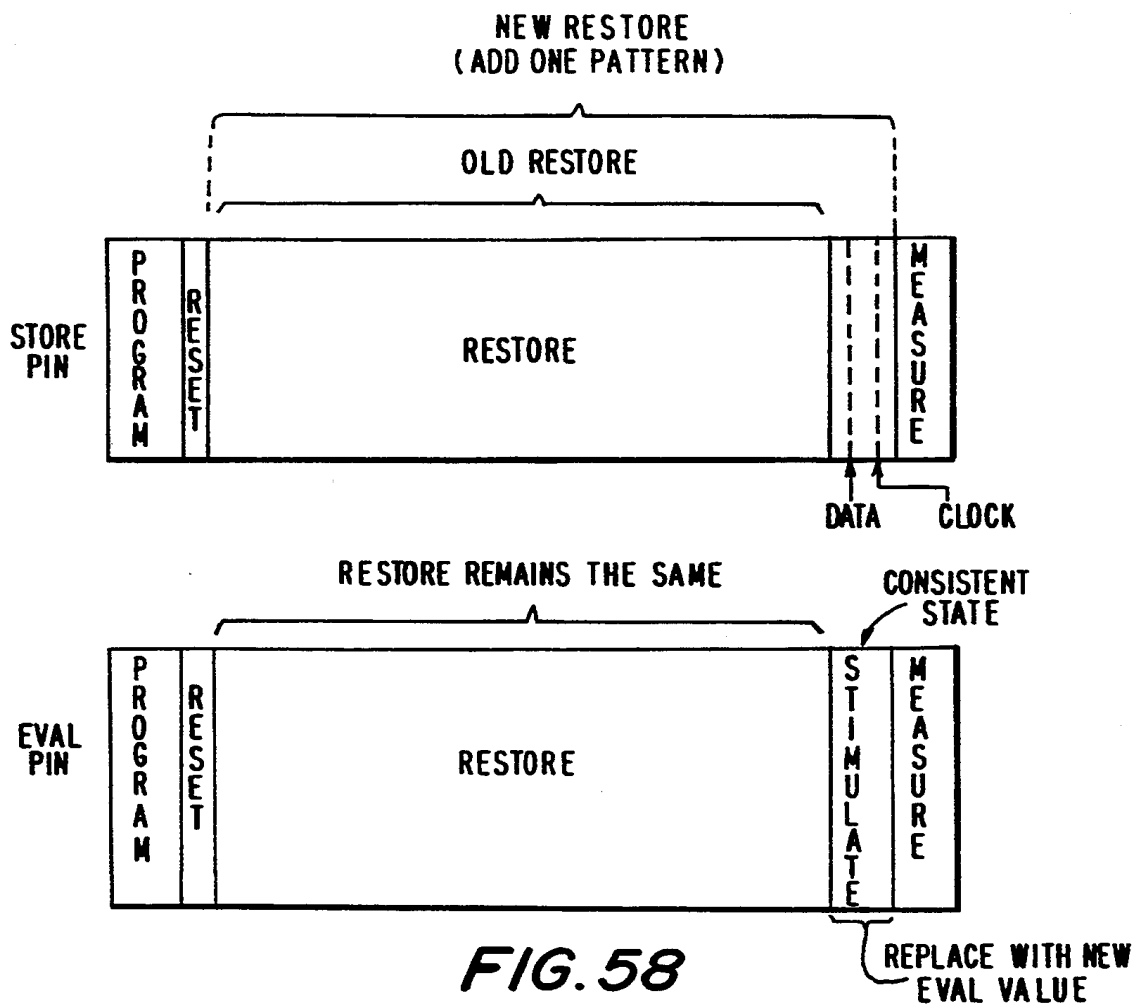


FIG. 58

HARDWARE MODELING SYSTEM AND METHOD OF USE

This is a continuation of application Ser. No. 07/359,711, filed May 31, 1989, now abandoned, entitled IMPROVED HARDWARE MODELING SYSTEM AND METHOD OF USE.

FIELD OF THE INVENTION

The present invention relates to systems used by electronics designers to simulate the operation of electronic circuits during development and testing of electronic systems. More specifically, the invention relates to hardware modeling systems that use examples of actual physical electronic circuits and components to model their behavior within a simulated electronic system design.

BACKGROUND OF THE INVENTION

A logic-simulation model of an electronic circuit is a diagnostic tool used by a logical-simulator to accurately mimic the logic and timing behavior of the circuit. The purpose of using such a model is to verify both logic and timing behavior of an operational electronic system containing the circuit. The internal operation and internal structure of a simulation model need not be similar to that of the actual circuit being simulated. However, the logical and timing behavior of the simulation model, as externally observed during simulation, must be identical to that of the actual circuit in its real environment.

Simulation models can be divided into three major classes: behavioral models, structural models, and hardware models. Behavioral models represent device behavior by using code written in an algorithmic language such as "C." They are compiled and linked with a simulator and run on a general-purpose computer. Structural models mimic the actual internal structure of a device; they generally run on either general-purpose or special-purpose computers. Behavioral models and structural models are both represented with software and are therefore also called "software models."

A hardware modeling system ("HMS") couples an actual "known-good" physical device (called a "hardware modeling element" or "HME") with a simulator; the physical device is used to model its own behavior during simulation. When the simulator requests a device evaluation, the HMS formats the inputs received from the simulator, applies the inputs to the physical device, measures device behavior, and returns the resulting outputs plus timing information to the simulator. The timing information is generally obtained from a table in a text file called a "software shell," which contains a representation of the data-book delays.

Before hardware modeling systems became available, logic simulation models were generally software models. Software models are practical for low-complexity circuits, but software models of complex very large scale integration ("VLSI") circuits have numerous disadvantages. For example, they may be costly and time consuming to develop, they may be inaccurate, they may be difficult to check, they may execute quite slowly, and they may not be portable between simulators.

In an attempt to address these problems, Valid Logic Systems in 1984 introduced the Realchip™ hardware modeling system which provided an alternative to software simulation models for complex circuits. Realchip

is a trademark of Valid Logic Systems. Valid Logic Systems' Realchip™ system was the basis of two U.S. Pat. Nos. 4,590,581, and 4,635,218, which issued in 1987.

Similar hardware modeling systems were subsequently introduced by Mentor Graphics ("HML"), Daisy Systems ("PMX"), GenRad ("HiChip"), Tera-dyne ("DataSource"), Cadnetix ("CDX-7000"), HHB Systems ("CATS"), and Tektronix ("TurboChip"). The Mentor Graphics HML hardware modeling system formed the basis for U.S. Pat. No. 4,744,084, which issued in 1988 and claimed improvements over the Real-chip™ system.

Users have attempted to apply these prior-art hardware modeling systems in several different ways in the design and test of electronic systems. The primary attempted application of prior-art hardware modeling systems is modeling standard devices and existing ASICs during simulation of electronic systems. In this mode, the designer may run real diagnostic software on the simulated system, verifying the operation of the entire system working together as if it had been prototyped.

Another attempted application of prior-art hardware modeling systems is to use an entire existing subsystem, such as a PC board, as an HME, verifying the operation of the entire system containing the PC board as if it had been prototyped.

Another attempted application of prior-art hardware modeling systems is to debug embedded system software and microcode before fabrication of the prototype system.

Another attempted application of prior-art hardware modeling systems is to examine the behavior of existing standard devices and ASICs for which specifications may be inadequate or inaccurate. In this mode, the design engineer uses the device in question as an HME, stimulating and observing it with the simulator's normal user interface. This enables him to directly and easily answer questions about its behavior while the system design progresses.

Another attempted application of prior-art hardware modeling systems is "in ASIC design. Many ASICs which are designed and verified in isolation do not operate properly within the target system. A hardware modeling system can help to solve this problem because it gives the designer access to full-function models of all the complex devices in the embedding system and enables him to simulate the entire system together before ASIC fabrication.

Another attempted application of prior-art hardware modeling systems is for functional verification of prototype ASICs. In this mode, the engineer fixtures the prototype ASIC as an HME, uses the resulting hardware model in place of the earlier gate-level model, and runs the same simulation test cases that were run during the design phase.

Although the prior-art hardware modeling systems enjoyed some commercial success, they suffer from a number of serious limitations with respect to these applications. One limitation of prior-art hardware modeling systems is that they lack the capability to be easily integrated with simulators from multiple different vendors. Therefore, in general, each one works with only one or at most a small set of simulators and cannot be accessed concurrently by multiple simulators running on multiple host platforms. Thus, designers using simulators from multiple vendors may need to buy multiple hardware modeling systems.

Also, prior-art hardware modeling systems are limited in their support for ASICs. For example, state-of-art ASICs have very high pin counts, and prior-art hardware modeling systems do not directly support very-high-pin-count devices. Also, prior-art hardware modeling systems do not directly support the capture and replay of test vectors for ASIC prototype verification.

Also, prior-art hardware modeling systems do not support direct timing measurement of HME output delays. Direct timing measurement is important for the comprehensive verification of prototype ASICs. Direct timing measurement is also important to simplify creation of the shell software for new ASIC hardware models.

Also, prior-art hardware modeling systems have proven difficult to use due to requirements for custom pin wiring on device adapters, specialized initialization circuitry, and complex software shells.

Also, prior-art hardware modeling systems are generally not fully and easily configurable. Hardware modeling systems can be used in several distinct modes which require configuration of the modeler in different dimensions. For example, system-level simulation requires that the HMS be configured for a large number of devices simultaneously mounted; ASICs tend to have high pin counts, so modeling and verifying them requires support for high-pin-count devices; and simulating the execution of software on processor models requires deep pattern memory. Prior-art hardware modeling systems limited simultaneous configurability in these dimensions.

Also, prior-art modeling systems suffer from poor performance, particularly in round-trip access time from the simulator, and in pattern-presentation rate. Better performance in both areas is necessary to keep pace with faster simulators and workstations, and to properly refresh state-of-the-art high-speed dynamic devices so that correct results are obtained.

Also, prior-art hardware modeling systems suffer from limitations of the "pin electronics," which is the electronic circuitry used by the hardware modeling system to stimulate and sample the inputs, outputs, and bi-directional ("I/O") pins of the HME. For example, because of the primitive nature of the pin electronics in the prior-art hardware modeling systems, when certain types of integrated circuits, particularly those having feed-through paths connecting I/O pins, such as the 74ALS645 bi-directional bus buffer manufactured by Texas Instruments, are used as HMEs in such systems, the very process of electronically sampling some I/O pins of the integrated circuit may inadvertently stimulate the device and cause incorrect signal values to be sampled on other I/O or output pins. As a result, the hardware modeling system may return incorrect results to the workstation which is performing the simulation. Some users of prior-art hardware modeling systems refer to this problem as the "coupled-I/Os problem."

Also, in prior-art hardware modeling systems, generally at least two bits are stored in each stimulus pattern for each pin of the HME, e.g., one bit for the data value and one bit which can inhibit the HMS pin driver. Since memory for pattern storage is generally expensive, a system and associated method of use is needed for reducing the amount and, therefore, the cost of required pattern storage.

Also, prior-art hardware modeling systems suffer from other deficiencies due to inadequate pin electron-

ics, such unreliability and a lack of capability to support multiple integrated circuit technologies (such as TTL and low-voltage CMOS). Because of the inability to support multiple integrated circuit technologies, some types of integrated circuits cannot be used as HMEs in prior-art hardware modeling systems.

Therefore, a need exists for an improved hardware modeling system, and a method which is directed toward solving and minimizing these and other problems of prior-art devices.

SUMMARY OF THE INVENTION

The present invention is an improved hardware modeling system and method of use.

The hardware modeling system ("HMS") of the present invention is adaptable for connection to a host computer, which includes a simulator, and is supported by software appropriate for communicating with the HMS. In fact, the HMS is capable of connecting to, and communicating with, a variety of host computers including a variety of simulators, in particular including conventional digital logic and fault simulators. The host computers are conventional electronic design workstations; the software that supports their use with the HMS is novel.

The host computer and the HMS are preferably connected by a local-area network over which information is transmitted bi-directionally between the two systems. Multiple HMSs can be networked with, and concurrently shared by, multiple host computers running multiple different simulators.

The HMS of the present invention provides hardware models of standard ICs, ASICs, and electronic subsystems. The HMS has a number of applications. Some of the major ones are as follows:

- 1) modeling standard off-the-shelf ICs, including static and dynamic ICs and VLSI ICs, in simulation of an electronic system design;
- 2) modeling custom static and dynamic ICs (ASICs) in simulation of an electronic system design;
- 3) modeling electronic subsystems and PC boards in simulation;
- 4) debugging embedded system software and micro-code using a simulated prototype;
- 5) examining the behavior of existing physical standard devices and ASICs;
- 6) verifying the functional behavior and timing of ASIC prototypes; and
- 7) generating and capturing standard-device, ASIC, and PC-board test vectors during simulation.

In order to better understand the present invention, it is useful to briefly describe the general overall operation of the HMS as it is used by a typical simulator.

When an electronic circuit is being simulated by a digital simulator including software simulation models of some circuit components, the simulator repeatedly "evaluates" the software simulation models of the components within the circuit. In order to evaluate a single software simulation model of a component, the simulator prepares a stimulus pattern representing the values of the input signals of the component and then executes a software routine which computes the output signal values of the component on the basis of both the stimulus pattern and the current internal state of the component. This output pattern is used by the simulator, along with other information, to prepare stimulus patterns for subsequent evaluations of the same and other components within the circuit. Methods of simulating elec-

tronic circuits using software simulation models of components are well known in the prior art.

The simulator may also access the HMS to evaluate the logical and timing behavior of an HME in response to simulated events on the inputs of a device instance modeled by the HME. Each access proceeds generally as follows:

The simulator first generates a stimulus pattern for the HME in generally the same manner as such a pattern is generated for a software simulation model. This stimulus pattern is encoded and transmitted to the HMS over a network. The HMS receives and decodes the stimulus pattern, stores the decoded pattern at the end of a "history sequence" of patterns in pattern memory, generally, as a single bit per pin of the HME, and presents the history sequence to the HME via the HMS pin electronics. The history sequence is appropriate to cause the HMS pin electronics to drive the input and I/O pins of the HME in such a way as to mimic the electrical behavior of the circuitry connected to the corresponding component in the circuit being simulated. Therefore, the physical HME behaves as if it were operating within the electronic circuit being simulated. After presentation of the history sequence, the pin electronics electronically samples the output and I/O signals from the HME. The HMS converts the sampled output and I/O signal values into a corresponding output pattern, which is encoded and returned via the network to the simulator. This output pattern is used by the simulator for further simulation in generally the same manner as an output pattern produced by a software simulation model.

The HMS hardware has two sections: control and modeler. The control section includes a central processing unit ("CPU") which runs the supporting software programs that perform all modeler functions and communicates with the simulators of the host computers. The CPU has local main memory. The modeler section contains special-purpose hardware to store pattern sequences, to present patterns to the HME, to sense its outputs, and to provide the outputs to the CPU. The modeler section includes a Pattern Bus, Timing Generator ("TG"), Pattern Controllers ("PACs"), Fast Pattern Memories ("PAMs"), Pin Electronics Modules ("PELs"), and Device Adapter Boards ("DABs") to which the HMEs connect.

The HMS can be independently configured in three dimensions: HME pin count, depth of pattern memory, and number of HMEs. The Pattern Bus (also referred to as the HMS "backplane") preferably includes four lanes. In the preferred embodiment, each lane supports HMEs with up to 80 pins. The TG, PACs, and PELs connect directly to the lanes. The CPU connects to the lanes through the TG. Each lane has its own pattern-presentation hardware that includes a PAC, preferably having from one to four PAMs connected to it. Each lane also has one to eight PELs connected to it. Each PEL can drive and sense up to 80 HME pins. Novel circuitry of the PAC, PAM, and PEL, and an associated method of use, allow the pattern memory in the PAMs connected to the PAC to be dynamically shared between pattern sequences for all HMEs in a lane of the PAC.

DABs of various sizes, but generally similar design and construction, connect to from one to four PELs in adjacent lanes. HMEs having from 241 to 320 pins can be mounted on a DAB connecting to four adjacent PELs, HMEs having from 161 to 240 pins can be

mounted on a DAB connecting to three adjacent PELs, HMEs having from 81 to 160 pins can be mounted on a DAB connecting to two adjacent PELs, and HMEs of 80 pins or less can be mounted on a DAB connecting to a single PEL. DABs of various sizes can be freely mixed in the HMS by connecting PELs at the appropriate sites at the backplane. For example, four 320-pin DABs can be mounted simultaneously with four 160-pin DABs and eight 80-pin DABs.

The TG, which connects to, and is controlled by, the CPU, is the main timing source for the HMS. The TG generates adjustable precision high-speed clocks and adjustable precision reference timing signals for timing the presentation of stimulus on individual HME pin drivers on a pin-by-pin basis and for timing the simultaneous sampling of HME outputs. The TG also synchronizes the Pattern Bus, checks for errors on the Pattern Bus, and provides a means for the CPU to access the Pattern Bus.

The Pattern Bus preferably is a high-speed controlled-impedance terminated bus supporting presentation of patterns for a 320-pin HME at up to 25 million ("25 M") patterns per second. The pattern bus operates in two modes: "access" and "presentation." In the access mode, the Pattern Bus provides the CPU with read/write access to all PACs, PAMs, PELs, and DABs in the HMS. In the presentation mode, the PACs drive pre-stored patterns from the Pattern Bus and present corresponding stimulus to the pins of an HME being used as a simulation model. PELs in adjacent lanes which are connected to a single DAB can simultaneously receive patterns in each lane. These patterns are presented by separate PACs in the separate lanes, which are synchronized by the TG. The HMS is thus able to present patterns to an HME having up to 320 pins and connected to up to four PELs at the same high pattern-presentation rate as for an HME having only 80 pins and connected to only one PEL.

The PAC and connected PAM are initialized by the CPU in access mode. The PAC contains the logic for accessing patterns in the PAMs mounted on it, error-checking the patterns, and delivering the patterns to PELs within its lane via the pattern bus. The PAMs connected to a single PAC can be used for storing a single pattern sequence to be presented to a single HME or for storing multiple pattern sequences to be presented to multiple HMEs. Pattern sequences consist of lists of pattern blocks linked by addresses stored in a link table memory on the PAC. The links can span multiple PAMs connected to a single PAC. Pattern blocks are dynamically allocated, released, and managed by the CPU, using linked-list management methods.

The PAM is organized as 102-bit words, each containing 80 pattern bits, eleven control bits, five spare bits, and six parity bits. Among other functions, the control bits select a target PEL within the lane, specify the function to be performed by the PEL, mark the last pattern in the pattern sequence, and specify a condition for branching to the link address for the pattern block. The PAM memory consists of two interleaved sets of video dynamic RAMs ("VRAMs"). To allow high-speed pattern presentation, alternate patterns are fetched from alternate sets. An entire sequence of linked pattern blocks filling all the PAMs on a PAC can be presented to the Pattern Bus at 25M patterns per second without pauses. Novel PAC and PAM circuitry and an associated method of use, allows the internal VRAM shift registers to be used to continuously repeat a pat-

tern sub-sequence to an HME while the HMS waits for a feedback condition to occur.

The PEL preferably includes control circuitry and five identical custom integrated circuits ("CICs"). A single PEL can support an HME having up to 80 pins. The PEL has programmable reference voltage generators and an electrical interface to the HME which allows it to support HMEs of any of a variety of logic families. Using precision timing signals from the TG, the PEL formats clocks and data, on a per-pin basis, to be presented to the HME. The PEL samples all of the HME pins and returns the sampled values for access by the CPU.

Novel circuitry and an associated method of use allow the HMS to accurately determine HME output delays. To determine HME output delays, the HMS repeatedly presents a fixed pattern sequence to the HME, each time sampling all HME outputs at a variable sampling time relative to the presentation time of the final pattern in the sequence. The HMS thus searches for the transition time of each HME output. A novel parallel binary search method reduces the number of repetitions required to determine all output transition times.

The five CICs on the PEL provide a means for driving and sensing the HME pins, and for checking for errors in the patterns received from the PAM via the PAC, Pattern Bus, and PEL. Each CIC preferably supports up to sixteen bi-directional HME pins. Functions of the CIC circuitry for driving and sensing HME pins (CIC "channels") are software programmable on a per-pin basis. Programming of each channel occurs serially using the Pattern Bus.

Novel analog and digital circuitry within the CIC and an associated method of use allow all pins of the HME, including I/O and control pins, to be accurately sampled while being simultaneously stimulated. This circuitry and method of use avoids any interference between the stimulus and the measurement by driving each I/O pin of the HME toward the target logic level of the simulated external circuitry using a current-limited soft driver in the CIC which can be overdriven to either logic level by the HME pin. At the same time that the CIC soft-drives the HME pin, the CIC compares the resulting voltage of the HME pin with four programmable reference voltages to determine whether the HME pin is driving toward the same logic level, the other logic level, is not driving, or is at an intermediate voltage level. Each channel of the CIC has the capability of distinguishing and retaining, for later CPU access, five states of each connected HME pin: driving low ("H0"), driving high ("H1"), non-driving low ("Z0"), non-driving high ("Z1") and unknown ("U"). In particular, the CIC channel has the capability to distinguish H0 from Z0 and H1 from Z1. Software-programmable reference voltages and software-programmable soft-drive current limits allow the CIC to properly stimulate and accurately sample HME devices of multiple logic families, including TTL, NMOS, CMOS, and low-voltage CMOS.

The DAB is the fixture for connecting the HME to be modeled to the PEL. The DAB has circuitry that allows the DAB, with the attached HME, to be connected to, and disconnected from, the HMS while the HMS is powered. Novel circuitry and an associated method of operation allow the HMS to automatically detect the presence and type of each DAB, and to track the DAB configuration as it changes. Novel connection

means included on the DAB allows a single DAB type to support devices with multiple different package styles, sizes, and types.

The HMS software consists of the core modeler code ("CMC") and the host resident code ("HRC"). Because of the novel structure of the CMC and HRC, the HMS can be quickly and easily integrated with multiple different simulators which run on multiple different workstation types.

The software that executes on the host computer includes the Simulator Function Interface ("SFI") and utilities. The SFI consists of a library of routines which are linked with a target simulator and provide access to the HMS. All communications between the simulator and the HMS is through the SFI. The utilities consist of a library of routines which provide HMS administration, remote hardware diagnostics, and test-vector support. The SFI and utilities are simulator independent and are written in a standard programming language ("C"). The SFI and utilities can be integrated with various simulators running on various workstation types without detailed knowledge of the CMC.

The CMC includes the complex HMS control software, including a real-time operating system, a network support module, a Simulator Function Interface command processor, a shell-software compiler, and a modeler handler. The CMC is simulator, and platform, independent.

In operation, the HMS provides a means for presenting stimulus events originating within the simulator to an HME serving as a simulation model, and for returning the correct results to the simulator.

Evaluation of a HME in response to a simulated event on an HME input proceeds as follows:

1. The HMS resets the HME to a known initial state using an "initialization sequence" specified in the shell software;
2. the HMS forces the HME to the internal state that the simulated device instance had prior to the current event;
3. the HMS presents the new event(s) on the appropriate HME pin(s);
4. after the HME outputs and I/Os have settled, the HMS samples all outputs and I/Os of the HME; and
5. the HMS returns any output changes to the simulator, with appropriate scheduling delays attached.

The scheduling delay for each output change is either inferred from a table of input-to-output delays in the shell software, or is directly measured by the HMS.

In general, forcing the state of the HME involves presenting a "history sequence" of patterns to the HME. The history sequence represents all events which have been presented by the simulator to the HME since the beginning of the simulation sequence. The HMS of the present invention uses novel pattern storage and presentation circuitry and method of use to represent the history sequence, generally using a single bit per pin in each pattern. The HMS of the present invention thus reduces the cost of required pattern memory and the cost of supporting pattern-transmission circuitry.

In general, for patterns in which a given HME I/O pin is not driving (as determined by previous measurement), the single pattern bit for the HME pin causes the CIC to drive the HME pin in the same direction that the simulated external circuitry is driving, and for patterns in which the HME pin itself is driving, the single bit causes the CIC to drive the HME pin in the same direction as the HME pin is driving. Patterns in which HME

I/O pins switch direction are represented and stored as two sequential patterns: the first enables or disables the CIC pin driver, as appropriate, and the second contains the logic level for the CIC to drive toward. The HMS drives HME I/O pins using the current-limited "medium driver" of the CIC. The current limit of the medium driver avoids excessive current levels when the CIC and HME drivers temporarily fight during switching. The HMS drives HME input pins using hard drivers, and drives HME output pins using soft drivers.

An object of the present invention is to provide an HMS and methods of use that will allow modeling and verification of standard off-the-shelf ICs, ASICs, and PC boards, including both functionality and timing for those devices.

Another object of the present invention is to provide an HMS and method of use that incorporates simulator-independent architecture with a standard, high-level interface.

Another object of the present invention is to provide an HMS and method of use that can be networked and concurrently accessed by multiple simulators of different types, running on multiple workstations or host computers of different types.

Another object of the present invention is to provide an HMS and method of use which provides shorter round-trip access times and provides higher pattern-presentation rates for very long pattern sequences.

Another object of the present invention is to provide an HMS that is more reliable than prior-art systems and that can be manufactured at lower cost for equivalent functionality.

Another object of the present invention is to provide an HMS and method of use that correctly determines the states of HME I/O pins independent of the manner in which the I/O pins are connected inside the HME.

Another object of the present invention is to provide an HMS and method of use which requires a smaller number of physical pattern memory bits to store the history sequences for HMEs.

Another object of the present invention is to provide an HMS which is independently and easily configurable in the dimensions of HME pins count, pattern-memory depth, and number of HME devices simultaneously mounted.

Another object of the present invention is to provide an HMS and method of use that supports HMEs of multiple logic families.

Another object of the present invention is to provide an HMS which is capable of more conveniently and economically supporting HMEs with very high pin counts.

Another object of the present invention is to provide an HMS and method of use that supports direct measurement of output delays for use in simulation and verification of prototype ASICs.

Another object of the present invention is to provide an HMS and method of use that will allow any member of a family of different HME package sizes and types to be fixtured using a single standard fixture.

Another object of the present invention is to provide an HMS and method of use that will allow HMEs to be fixtured without extensive custom wiring, special initialization circuitry, or complex software shells.

Another object of the present invention is to provide an HMS and method of use that permits an HME to be connected to, or disconnected from the HMS while the HMS is powered.

Another object of the present invention is to provide an HMS and method of use that will allow the collection, in a simulator-independent manner, of test vectors for an HME being used in a simulation.

Another object of the present invention is to provide an HMS and method of use that will allow the functional verification of HME operation relative to a set of test vectors in a simulator-independent manner.

These and other objects of the present invention will be described more fully in the remainder of the specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic drawing of the preferred embodiment of the HMS of the present invention.

FIG. 2 is a block diagram of a network of workstations and the HMSs of the present invention.

FIG. 3 shows the categories of functions supported by the Simulator Function Interface of the present invention.

FIG. 4 is a block diagram of the Utilities.

FIG. 5 is a block diagram of the Shell Software.

FIG. 6 is a block diagram of the Core Modeler Code.

FIG. 7 is a block diagram of the TG of the HMS of FIG. 1.

FIG. 8 is a block diagram of a PAC and attached PAMs of the HMS of FIG. 1.

FIG. 9 is a schematic drawing of the TMG/CPU Interface of the PAC of FIG. 8.

FIG. 10 is a schematic drawing of the memory control circuitry of the PAC of FIG. 8.

FIG. 11 is a schematic drawing of the refresh control circuitry of the PAC of FIG. 8.

FIGS. 12 and 13 are schematic drawings of the pattern control circuitry of the PAC of FIG. 8.

FIG. 14 is a schematic drawing of the link table circuitry of the PAC of FIG. 8.

FIG. 15 is a schematic drawing of the PAM detection circuitry of the PAC of FIG. 8.

FIG. 16 is a schematic drawing of the Configuration PROM and associated circuitry of the PAC of FIG. 8.

FIG. 17 is a block diagram of the 102-bit pattern word according to the present invention.

FIG. 18 shows bit assignments of the pattern control word according to the present invention.

FIG. 19 is a schematic drawing of the preferred embodiment of the control circuitry for the PAM of the present invention.

FIG. 20 is a block diagram of a pattern sequence according to the present invention.

FIG. 21 is schematic drawing of the preferred embodiment of the PEL circuitry which interfaces the Pattern Bus of the HMS of FIG. 1 with an HME.

FIGS. 22-24 are schematic drawings of the control portion of the PEL circuitry of the HMS of FIG. 1.

FIG. 25 is a block diagram of a CIC on the PEL circuitry of the present invention.

FIG. 26 is a block diagram showing the parallel connection of a number of CICs of FIG. 25.

FIG. 27 is a block diagram illustrating the serial connection of a number of CICs of FIG. 25.

FIG. 28 is a schematic drawing of the digital circuitry of the CIC of FIG. 25.

FIG. 29 is a schematic drawing of the Parity Checker circuitry of the CIC of FIG. 25.

FIG. 30 is a schematic drawing of the Error Handler circuitry of the CIC of FIG. 25.

FIG. 31 is a schematic drawing of the Control Circuit of the CIC of FIG. 25.

FIG. 32 is a schematic drawing of the Data Channel of the CIC of FIG. 25.

FIGS. 33 and 34 are a schematic drawings of the Data and Control Register of the Data Channel of FIG. 32.

FIG. 35 is a schematic drawing of the Edge Selector circuitry the CIC of FIG. 25.

FIG. 36 is a schematic drawing of the Formatter/Driver circuitry of the Data Channel of FIG. 32.

FIG. 37 is a schematic drawing of the Hard-Drive Output Enable Formatter of the Formatter/Driver circuitry of FIG. 36.

FIG. 38 is a schematic drawing of the Medium-Drive Output Enable Formatter of the Formatter/Driver circuitry of FIG. 36.

FIG. 39 is a schematic drawing of the Data Formatter of the Formatter/Driver circuitry of FIG. 36.

FIG. 40 shows the four signal formats for transmitting data and clocks to the HME in accordance with the present invention.

FIG. 41 is a schematic drawing of the Soft-Drive Output Enable Formatter of the Formatter/Driver circuitry of FIG. 36.

FIG. 42 is a schematic drawing of the sampling circuitry of the Data Channel of FIG. 32.

FIG. 43 is a schematic drawing of the Short Sensor circuitry of the sampling circuitry of FIG. 42.

FIG. 44 is a block diagram of the analog sections of the CIC of FIG. 25.

FIGS. 45 and 46 are current/voltage curves for the medium driver of FIG. 47.

FIG. 47 is a schematic drawing of the soft, medium, and hard drivers of the CIC of FIG. 25.

FIGS. 48A-48E show a representation of the I/V characteristic of the low and high soft drivers of FIG. 47.

FIG. 49 is a schematic drawing of the 5-State Sampler of the sampling circuitry of FIG. 42.

FIG. 50 shows a representation of the voltage ranges for the 5-state sampling of a TTL HME device.

FIG. 51 is a top view of a representative DAB of the HMS of FIG. 1.

FIGS. 52A-52F shows representative HME footprints to accommodate attachment of Pin Grid Array IC packages to the DAB of FIG. 51.

FIG. 53 is a schematic drawing of the circuit for generating variable HMEVCC voltages of the DAB of FIG. 51.

FIGS. 54A and 54B shows representative trace cuts in the configuration area of the DAB of FIG. 51.

FIG. 55 shows a representation of the timing measurement process performed in accordance with the present invention.

FIG. 56 is a schematic drawing of a representative circuit containing an I/O store pin.

FIG. 57 shows a representative method of system operation using two bits per pin to sample an I/O store pin.

FIG. 58 is a diagram of the method for evaluating the store and eval pins of an HME.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

The present invention is an improved HMS and method of use which may be embodied as a stand-alone system for connection to one or a variety of host com-

puters, called workstations, that are used to design digital electronic systems.

FIG. 1 shows the HMS of the present invention generally at 100. The HMS includes the section associated with the host computer, shown generally at 112, and the stand-alone section, shown generally at 110. The host computer and the stand-alone section of the HMS are preferably connected by an Ethernet® (IEEE 802.3), which is shown at 116. Ethernet® is a registered trademark of Xerox Corporation. Preferably, the Ethernet® hardware and software systems manufactured by Sun Microsystems are used. The Ethernet® protocols that are preferably used are the TCP/IP and UDP/IP protocols. The use of Ethernet® allows networking of a number of HMSs and host computers.

After a brief overview of the HMS of the present invention, the section associated with the host computer will be discussed in detail, then the stand-alone section will be discussed in detail.

The HMS 100 can be dedicated to a particular host computer, or it can be part of a network as shown in FIG. 2. The network generally shown at 180 includes HMSs, HMS1 to HMSN, and representative network workstations 186, 188, 190, 192, 194, 196, and 198. In network 180, the HMSs support multiple simulations on the several different workstations. Different simulator programs can run on the different workstations and can all concurrently access any HME on any of the HMSs.

The host computers shown in FIG. 2 are conventional workstations. They are representative host computers to which the HMS of the present invention may be connected. These host computers are the Sun-3™ and Sun-4™ manufactured by Sun Microsystems; Apollo® DN-3000, Apollo® DN-4000, and Apollo® DN-10000 manufactured by Apollo Computer; 80286-based and 80386-based personal computers manufactured by Compaq and IBM; and DEC VAX™ computers manufactured by Digital Equipment Corporation. Sun-3™ and Sun-4™ are trademarks of Sun Microsystems, Inc.; Apollo® is a registered trademark of Apollo Computer, Inc.; and DEC VAX™ is a trademark of Digital Equipment Corporation.

Again referring to FIG. 1, the depiction at 112 represents the host computer and the novel host resident software and shell software, which are part of the HMS of the present invention. Host operating system 160 and logic/fault simulator 164 are part of the conventional host computer and are not part of the present invention.

A listing of all the HMS software, including the CMC software, is attached as Appendix III. This listing of HMS software is written in the "C" programming language, as supported by the Sun-3™ workstation manufactured by Sun Microsystems. The software listing is divided into major sections: "lm," "include," "sfi," "networkh," "networkm," "networkc," "shellsw," "shellswm," "shellswc," "diags," "bsp.diags," "tasks.diags," "lm1000," "bsp.lm1000," "tasks.lm1000," "os," and "misc." The name of the major section is shown at the top of each listing page in the box labeled "SOURCE PROGRAM." The "SOURCE PROGRAM" box also shows the name of the routine listing on the page, following the major section name. Where appropriate, the specification will refer to specific portions of supporting software.

In order to better understand the HMS software, each of the major software sections will be briefly described. The "lm" section contains the primary source code for the utilities. This code relies on the "sfi" sec-

tion for low level routines. The "include" section contains the source code for all "C" language include files that are shared by more than one section of code. The "sfi" section contains the source code for communication between the simulator and the HMS. The "networkh" section contains the source code that implements the host side of the Ethernet®/UDP/IP support. The "networkm" section contains the source code that implements the upper levels of the modeler side of the Ethernet®/UDP/IP support. The lower levels of the network support are part of the VRTX® operating system. VRTX® is a registered trademark of Ready Systems. The "networkc" section contains the source code for routines that are common to both the host and HMS sides of the network support. The "shellsw" section contains the source code for the small amount of shell software processing that is performed on the host. The "shellswm" section contains the source code for the shell software processing that is performed on the HMS: all scanning, parsing, constraining, and back-end code generation of the shell software is performed by this code. The "shellswc" section contains the source code for routines that are common to both the host and HMS sides of the shell software processing. The "diags" section contains the source code implementing the HMS diagnostics. The "bsp.diags" section contains low-level source code that supports interrupts, timers, UARTS, etc., during execution of the diagnostics on the HMS. The "tasks.diags" section contains the source code that implements the VRTX® tasking model used during execution of the diagnostics; for example, there is a housekeeping task that executes as the lowest priority, performing various watchdog and clean-up activities as needed, there are receive and transmit tasks that handle modem and console port communications. The "lm1000" section contains the source code that implements pattern management, HME evaluation, and other basic HMS functionality. The "bsp.lm1000" section contains the source code that supports interrupts, timers, UARTS, etc., during the execution of the CMC. The "tasks.lm1000" section contains the source code that implements the VRTX® tasking model used during execution of the CMC. The "os" section contains the source code that implements the VRTX® support, including low-level network control and control of all CPU on-board resources. The "misc" section contains the source code implementing miscellaneous functions common to both the host and HMS code.

The host resident software includes SFI 162 and utilities 166. Shell software 114 is not part of the host resident software. SFI 162 is a library of routines which are linked with logic/fault simulator 164 and provide the simulator access to the section of the HMS shown at 100. The SFI consists of the routines and definitions in the major software listing sections "sfi," "include," "networkh," "networkc," "shellsw," "shellswc," and "misc." The SFI integrates eighteen HMS functions into simulator code, giving the simulator the ability to create HME definitions, instances, and faults on the HMS. The SFI also enables the simulator to map the simulator pin names to HMS pin names and map simulator logic/strength values to and from values supported by the HMS. The SFI permits input and I/O pin values from the simulator to be presented to the HME, and permits the sensed logic/strength value of an output pin or I/O pin transition to be returned to the simulator, with the minimum/typical/maximun delay values.

Preferably, SFI 162 is written in the "C" programming language as supported by the Sun-3™ workstation manufactured by Sun Microsystems. If the simulator is written in a language other than "C," the simulator and HMS may be linked by routines that bridge the simulator language to "C." This feature, coupled with other features, renders the SFI independent of the simulator and host computer.

The SFI routines are divided into several categories on the basis of their use. These categories are shown in FIG. 3 generally at 200. The functions of Begin/End category 202 are used to begin an HMS modeling session and to end the session. The `lm_begin_modeling_session` function is used at the beginning of a simulation to allocate all host data structures, to initialize the communications link, and to allocate and initialize all HMS resources. The `lm_end_modeling_session` procedure is used to terminate the session and release all host and HMS resources. Each of these functions is called once per simulation.

The functions in Create category 204 are used to create device definitions (`lm_create_definition`), instances of these defined devices (`lm_create_instance`), and instances of the defined devices in faulty circuits (`lm_create_fault`). These three create functions generate a tree structure with the device definition as the root, device instances at the next level, and faulty circuit device instances (faults) at the leaf level. The fault creation functions are normally only used for fault simulators.

The functions in Release category 206 include the functions `lm_release_definition`, `lm_release_instance`, and `lm_release_fault`. These three functions are the complements of the three create functions, and are used to release (free) the resources associated with their respective entities.

The functions in Set category 208 establish settable parameters, mappings, and values. The function `lm_set_simulation_tick` defines the simulator's minimum time unit, and the function `lm_set_simulation_time` establishes the current simulation duration (in simulation ticks). Following a call to the function, `lm_set_simulation_time` establishes the current simulation duration (in simulation ticks). Following a call to the function `lm_set_simulation_tick`, all time information communicated to or from the simulator is in the defined time units. The function `lm_set_pin_map` creates a 1-to-1 mapping from simulator pin identifiers to HMS pin names defined in the shell software. The function `lm_set_logic_level_map` creates a mapping from simulator logic/strength values to those supported by the HMS. This mapping allows the simulator to use any desired internal representation to communicate with the HMS. The function `lm_set_pin_value` then allows the simulator to establish the logic/strength values for both input and I/O pins using the predefined logic level map. Only signal transitions need be communicated to the HMS, increasing performance and reducing network traffic.

Get category 210 contains only the function `lm_get_pin_value`. This function is used to retrieve the logic/strength value of an output (or I/O) pin, and the minimum, maximum, and typical delay values for the transition. Only outputs that transition are returned from the HMS.

Save/Restore category 212 contains two functions. The function `lm_save_modeler_state` saves the current state of the modeling session to a file, and the func-

tion `lm_restore_modeler_state` restores that state. This pair of functions may be used to allow suspension of a simulation for later resumption. The simulation state is stored as a file which is host and HMS-independent; any HMS which contains the necessary resources may be used for restoring. Thus, a save can be done on one HMS and the state can be restored on a different HMS.

The functions in Inquire category 214 provide a general mechanism for checking the status of the HMS parameters. These parameters include the total pattern memory, total system memory, available pattern memory, number of users, number of lanes and slots, number of HMEs, number of PELs, number of active instances, number of active faults, type and version number of each system component, available devices and their locations in the system, types of HMEs in the system and their DAB IDs, memory usage in each lane, memory usage of each instance, and memory usage of each user.

The functions in Miscellaneous category 216 perform the following operations: selecting a particular HMS for inquires, turning timing measurement on and off, extracting shell software created by the Create category, setting the HMS to repeat a specific pattern history in an infinite loop to allow observation of the HME using external test equipment, and conducting maintenance and diagnostics.

Utilities 166 is a library of routines which are installed on the host workstation and provide menu-driven simulator-independent utility programs designed to help create and verify models, perform maintenance on the HMS, and perform diagnostics. Utilities 166 consists of the routines and definitions in the major software listing sections "lm," "sfi," "include," "networkh," "networkc," "shellswl," "shellswc," and "misc." The Utilities have five main categories, each of which has a submenu. The five main categories are Create Logic Models category 222, Verify Logic Models category 224, Perform Maintenance category 226, Run Diagnostics category 228, and Show Modeler Configuration category 230.

Create Logic Models category 222 has the submenu shown at 231. The submenu includes Label Device Adapter and Check Shell Software. The Label Device Adapter labels the DAB. The label must match the label given in the Shell Software. The Check Shell Software utility performs a syntactic and semantic check on the specified shell software by sending the shell software to the HMS, where it is parsed and error checked.

Verify Logic Models category 224 has the submenu shown at 232. The submenu includes Play Vectors, Measure Timing, and Create Timing File. The Play Vectors utility presents test patterns in TSSI format, as supported by Test System Strategies, to the HME and compares the results with the expected results to verify the correct operation of the HME. Specifically, this utility creates an instance of the HME, plays patterns from a Test Vector file to the HME, and measures the states of the output and I/O pins of the HME after each pattern. It also stores the results in an output file. If there is a discrepancy, the instance number, pattern number within the file, and the pattern time are provided to the user.

The Measure Timing utility measures the actual physical output delays of the HME. This utility repeatedly presents inputs from the Test Vector input file to the

HME, measures the resulting delays, and stores them in the Timing Measurement file.

The Create Timing File utility creates a single Timing file from one or more Timing Measurement files that were previously created with the Measure Timing utility. The Create Timing File utility reads all specified files, and for each delay, records the minimum and maximum values, and calculates the typical delay by averaging the minimum and maximum values. The resulting file lists each delay only once, and is appropriate for use as a shell software Timing file for the HMS.

Perform Maintenance category 226 has the submenu shown at 233. The submenu includes Install Modeler, Boot Modeler, Shut Down Modeler, Reboot Modeler, Set Baud Rates, Set Idle Process Timeout, and Abort User. The Install Modeler utility configures the HMS for booting. It collects and sends the boot information to the HMS, including down loading the CMC.

The Boot Modeler utility manually boots the HMS. In order to boot the HMS, the HMS must be shut down, and must wait for the CMC to be down loaded.

The Shut Down Modeler utility performs an orderly shutdown of the HMS. The HMS will remain shut down until it is booted with the Boot Modeler utility.

The Reboot Modeler utility performs an orderly shutdown of the HMS and immediately restarts the HMS.

The Set Baud Rates utility allows the user to set the baud rates of the two RS-232 HMS communication ports. The new baud rates are effective immediately after executing the utility.

The Set Idle Process Timeout utility establishes a timeout period on the HMS. When the timeout period is set, any process that has remained idle for the specified time period on the HMS will be checked by the host computer, and if the connected process no longer exists on the host computer, the modeling session is terminated and resources that had been allocated to the user on the HMS are made available to other users.

The Abort User utility allows the user to manually remove the HMS processes which are no longer active. Inactive processes are left on the HMS when the simulation session or utilities session terminates abnormally. Inactive processes may be holding resources on the HMS which are needed for other simulation sessions. The Abort User utility releases all HMS resources associated with a particular user.

Run Diagnostics category 228 has the submenu shown at 234. It includes Execute All Tests, Diagnostic Adapter Menu, CPU Menu, Timing Generator Menu, Pattern Controller and Pin Electronics Menu (per lane), Multi-Lane Menu, and External Clock Menu. The Run Diagnostics category allows the user to run HMS diagnostic programs from the host computer.

The Execute All Tests utility runs all of the diagnostic tests. As each test is completed, a status message is generated which is representative of the test that was performed and the outcome of the test.

The Diagnostic Adapter Tests utility performs tests on the PEL using a special diagnostic adapter which connects to the PEL in place of an HME. The diagnostic adapter plugs into a single PEL and allows all of the pin drivers and sensors of the PEL to be exercised using a signature analysis algorithm. The design of the diagnostic adapter is well within the capability of one skilled in the art. To fully verify the PELs, this utility should be run for each PEL. When this utility is used, the diagnostics search every configured lane in the HMS

and perform the tests on every PEL that has a diagnostic adapter installed.

The CPU Menu, Timing Generator Menu, Pattern Controller Menu and Pin Electronics Menu (per lane), and Multi-Lane Menu contain the detailed diagnostic testing utilities for the specified component or circuitry.

The External Clock Menu utilities measure and verify the frequency of external clocks connected to the two external clock connections of the HMS. These tests are not run as part of the Execute All Tests utility because this utility requires additional hardware that may not always be connected to the HMS.

Show Modeler Configuration category 230 has the submenu shown at 235. This submenu includes the utilities Show Modelers, Show Logic Models, Show Users, and Show Versions. The Show Modelers utility lists the available HMSs on the network and describes their configuration. The configuration information may include a list of major hardware components, the amount of CPU memory being used, the number and location of PACs, PAMs, or PELs, the amount of pattern memory used or available per lane, the number of instances and faults being modeled per lane, a list of active users and their host machines, and the software version being used.

The Show Logic Models utility lists the HMEs currently available on a specified HMS, and provides information about each HME. This information may include its lane/slot location, the HME status, whether it is used as a public or private device, the number of active instances of the HME, the number of active faults, the type of DAB, and the DAB revision number and manufacturer.

The Show Users utility provides a brief summary of the status information about each current user. Such information may include the host name, the total number of public or private HMEs used, the total amount of pattern memory used and the total number of active instances and active faults created by the user.

The Show Versions utility provides the current revision levels of all installed HMS hardware and software.

FIG. 5 shows the structure of shell software 114. By way of example, a listing of preferred shell software for an HME containing the MC68020 microprocessor manufactured by Motorola Semiconductor and a listing of preferred shell software for an HME containing the AM7990 Ethernet® controller IC manufactured by Advanced Micro Devices are provided in Appendices I and II, respectively. The shell software describes the HME to the HMS. The exact contents of the shell software varies depending upon the characteristics of the HME device. The following describes the features and capabilities of shell software in general.

All processing of the shell software is done on the HMS. The shell software consists of statements arranged in files that define the specific characteristics of the HME to be modeled, and the DAB to which it is attached. Table 1 lists representative statements, the file type, and a brief description of the shell software.

TABLE 1

Statement	File	Description
device_name	Model	Associates the shell software with a particular HME.
technology	Device	Specifies the device technology process.

TABLE 1-continued

Statement	File	Description
device_speed	Device	Specifies the device clock frequency range.
in_pin	Device	Defines the input pins and attributes.
out_pin	Device	Defines the output pins and attributes.
io_pin	Device	Defines the I/O pins and attributes.
power_pin	Device	Defines the power pin numbers.
ground_pin	Device	Defines the ground pin numbers.
nc_pin	Device	Defines the no-connect pin numbers.
package_mapping	Package	Defines the device package mapping.
adapter_mapping	Adapter	Defines the DAB mapping.
modeler_name	Model	Selects a specific HMS.
device_usage	Device	Enables and disables pattern history replay.
clock_type	Device	Selects the source of the pattern clock.
device_setup_time	Device	Specifies the worst case setup time of any HME pin.
device_hold_time	Device	Specifies the worst case hold time of any HME pin.
initialize	Device	Defines the initialization pattern sequence.
delay	Timing	Lists the data sheet propagation delays.

The shell software for an HME and DAB may be stored in a single file, except for the test vectors, which are stored in a separate file. However, the preferred file structure of shell software 114 is shown in FIG. 5. In FIG. 5, main Model file 236 serves as the header file for the shell software of the HME. The Model file contains the "device_name" statement, the "modeler_name" statement, and a list of shell software file directives which will be explained. The required "device_name" statement is used to associate the shell software with the physical DAB and HME. The character string "name" specified in the statement must exactly match the character string used to label the DAB as per the Label Device Adapter utility. If the two do not match, the HMS will not be able to locate the proper hardware for the shell software. The optional "modeler_name" statement is used to specify the name of the HMS to be used in case there are multiple HMSs on the network.

The four directives "physical," "timing," "package," and "adapter," provide a preferred method of structuring the shell software. The "physical" directive is used to include the information in the Device file 237 that describes the HME being modeled. The "timing" directive is used to include the information in Timing file 238 regarding model timing. The "package" directive is used to include the information in Package file 239 that describes the HME package. The "adapter" directive is used to include the information in Adapter File 241 that describes the DAB being used.

The statements of Device file 237 will now be described. The required "technology" statement describes the power supply and current levels appropriate for the fabrication technology of the HME, which could be, for example, NMOS, CMOS, LCMOS, or TTL. The current values defined with the "technology" statement apply to all pins of the HME, but may be overridden on a per-pin basis using the "out_pin" and "in_pin" statements.

The required "device_speed" statement specifies the allowable frequency or period of the pattern clock of the HME. The frequency that is specified is either a single maximum or a range. The specified period may also be either a single maximum or a range.

The six HME pin description statements are "in_pin," "out_pin," "io_pin," "power_pin," "ground_pin," and "nc_pin." All HME pins should be listed in the shell software as one of these pin types. Input and I/O pins have simulator data presented to them, while output and I/O pins are sampled to detect changes, which are then passed back to the simulator. Power, ground, and no-connect pins are not driven during simulation.

The "in_pin" statement maps the user-defined input pin name to the actual HME package pin number. The input pin attributes, which are part of the statement, specify which inputs can affect the state and the outputs of the HME. The attributes for "in_pin" are "eval," "store," "edge_rise," "edge_fall," and "keepalive." Changes on an "eval" pin may cause changes at the outputs of the HME, but do not cause changes in the internal state of the HME. Any rising or falling changes on a "store" pin may cause changes at the outputs of the HME and changes in the internal state of the HME. An "edge_rise" pin is a pin with respect to which only a rising edge on the pin may cause changes to the outputs and internal state of the HME. An "edge_fall" pin is a pin with respect to which only a falling edge on the pin may cause changes to the outputs and internal state of the HME. The "keepalive" attribute should be included to document which pin has the keepalive signal connected to it on the DAB. The "keepalive" attribute is used in addition to any of the other attributes.

The "out_pin" statement maps the user-defined output pin name to the actual HME package pin number. One output pin attribute is "feedback." If the HME pin is being used for feedback in an initialization sequence, then the "feedback" attribute must be specified. In this case, the HME pin must be connected to the feedback pad on the DAB, and the feedback sequence must be described in the "initialize" statement.

The "io_pin" statement maps the user defined I/O pin name to the actual HME package pin number. I/O pins share all of the attributes of input and output pins.

The "power_pin," "ground_pin," and "nc_pin" statements define the HME package pin numbers for the power, ground, and no-connect pins, respectively. These pins are not driven during simulation.

The optional "device_usage" statement is used to indicate whether the HME is public or private. A public HME is one to which a history sequence is presented in order to restore its internal state. A private HME retains its state internally between evaluations. A public HME can be shared between multiple simulations and between multiple device instances in a simulated design, whereas, a private HME can represent only a single device instance in a single simulation. Only static devices (devices with no minimum clock-rate specifica-

tion) can be used in private HMEs. Either static or dynamic devices can be used in public HMEs.

The optional "clock-type" statement specifies the clock source for the HME. The clock may be the internal clock signal generated by the TG, or it may be an external clock.

The optional "device_setup_time" statement is for overriding the default setup time. The HMS will always present input and I/O transitions at least as much in advance of HME store, edge_rise, and edge_fall pin transitions as this parameter specifies, thus guaranteeing at least this amount of setup time.

The optional "device_hold_time" statement is for overriding the default hold time.

The optional "initialize" statement is used whenever a public device is evaluated, to put the device in the same fixed internal state prior to presenting the history sequence. This statement defines an initialization sequence that is applied to the HME for this purpose. This may be accomplished by an initialization sequence consisting of fixed patterns, or by feedback initialization, which consists of a pattern sequence that is repeatedly presented to the HME until a desired internal state is attained. The two types of sequences will be discussed in detail, subsequently.

Timing file 238 includes one statement, the "delay" statement. The HMS supports minimum/typical/maximum HME output delays by means of the shell software "delay" statement. When delay information about an output or I/O pin is provided in the Timing file, the value specified for the delay is returned to the simulator whenever that pin changes state during simulation.

Package file 239 and Adapter file 241 specify the mapping between the HME package pin numbers and the DAB net numbers which connect to the PEL. The "package_mapping" statement reflects the design of the HME device package; it translates HME device package pin numbers into DAB pin names. The "adapter_mapping" statement reflects the design of the DAB itself; it translates DAB pin names into DAB net numbers. Every DAB pin name in the master footprint area of the DAB maps to one DAB net number. Multiple DAB pin names may map to a single DAB net number.

Test Vector file 243 contains input stimuli and expected outputs for the HME in TSSI format, as supported by Test System Strategies. This file constitutes a functional test program for the HME.

Having discussed the section of the HMS associated with the host computer, stand-alone section 110 and its elements will now be discussed. Before describing the hardware elements of stand alone section 110, the software that executes on the CPU of the HMS will be described. This software is the CMC.

Referring to FIG. 6, the CMC will be disclosed. The CMC includes diagnostics and run-time code that execute on CPU 240 of the HMS. The CMC diagnostics are used to verify the correct operation of the HMS hardware. They are downloaded to stand alone section 110 when the user runs diagnostics. The CMC diagnostics includes the routines and definitions in the major software listing sections "diags," "networkm," "networkc," "shellswm," "shellswc," "misc," "bsp.diags," and "tasks.diags." The CMC run-time code controls the normal operation of stand alone section 110. It manages pattern sequences in pattern memory, controls HME evaluations, manages the network, compiles shell software, and interfaces to the real-time operating system. The CMC run-time code includes the routines and defi-

nitions in the major software listing sections "lm1000," "networkm," "networkc," "shellswm," "shellswc," "misc," "bsp. lm1000," and "tasks.lm1000."

The CMC files preferably are ASCII files. These files are downloaded from the host computer over the Ethernet network to the HMS when the HMS is booted. The CMC includes the Ethernet® support software for the modeler, as well as the modeler control software. Most of the operations required to service simulator requests are performed by the CMC on the CPU.

The CMC software is stored on the host's mass storage system. After power-up or reset, the CMC diagnostics are downloaded to the CPU and executed. If the CMS diagnostics are executed successfully, the CMC run-time code is downloaded and executed. The CMC automatically determines the configuration of the PAMs, PELs, and the DABs. If a valid configuration is found, the CMC waits for requests from the host computer.

Referring now to FIG. 6, the execution of the CMC is supported by a real-time operating system 250. In the preferred embodiment it is the VRTX® system that is commercially available from Ready Systems. The VRTX® system manages the hardware resources of the CPU. Layered on top of the operating system is network support 252 for the Ethernet® connection protocols that are needed to communicate with the host system. The network layer creates a packetized byte stream between the host and the HMS. All communications between the host and the HMS are by this byte stream. The SFI command processing layer 254 on the HMS receives requests from the host computer and a byte stream provided by network layer 252. The SFI command processing layer processes host requests and generates replies which are passed back via the network to the requesting host.

SFI command processing calls upon two major subsections of the CMC to answer host requests. These are language processing and pattern handling subsections. Language processing module 258 parses and error checks the shell software and compiles a binary representation which is used by the HMS for the duration of the simulation. Pattern handling module 256 processes simulator requests for HME evaluation, and returns sampled HME pin changes and output delays to SFI layer 254 for delivery to the host via the network.

Multiple host and multiple user support preferably is provided by queuing multiple requests and then scheduling the CMC to handle each request on a first-come, first-serve basis.

Stand-alone section 110 forms the majority of the HMS of the present invention. For convenience, unless specified differently, a reference to 110 is a reference to the HMS generally.

Referring again to FIG. 1, the HMS at 110 will be described. The elements that are part of the stand-alone section 110 of HMS 100 and their interconnection will be first described in general, then each will be described in greater detail, and the supporting software will be described.

The HMS has two sections: control and modeler. The control section includes CPU 240. The modeler section includes TG 248, the Pattern Bus at 250, 252, 254, and 256, the PACs at 258, 260, 262, and 264, the PAMs at 259, 261, 263, and 265, the representative PEL 266, and the DABs 268, 270, 272, and 274.

CPU 240 is connected to Ethernet® 116 at node 118 by line 242. The CPU preferably has 4 Mbytes of memory. The CPU runs the CMC software and provides Ethernet® network support. TG 248 is connected to CPU 240. TG 248 generates precision timing edges and synchronizes the playing of patterns. CPU 240 accesses the remainder of the HMS through the TG. TG 248 is connected to the four lanes of the Pattern Bus at 250, 252, 254, and 256. The Pattern Bus is used to distribute data in both access and pattern-presentation modes of the HMS. In access mode, the lanes of the Pattern Bus are used to provide the CPU with access to the HMS components.

The HMS preferably has one to four PACs, which are shown at 258, 260, 262, and 264. Each PAC is connected to one of the Pattern Bus lanes. The PAC contains the logic for pattern sequencing. It delivers patterns from the PAMs onto the Pattern Bus for delivery to the PELs. Each PAC has at least one PAM connected to it. Up to four PAMs may be connected to each PAC. The maximum number of patterns stored in the PAMs attached to a single PAC is 2M (512K in each PAM).

PEL 266 is one of up to 32 PELs of the HMS. Each lane may have PELs connected to it. The PELs provide inputs and sense outputs of the HME. The PELs include custom ICs ("CICs") that are novel. The HME interfaces with the PEL via the DAB. DABs 268, 270, 272, and 274 connect to different PELs. The number of HME pins determines the number of PELs to which to the DAB must connect and, thereby, the DAB type. DAB 268 supports HMEs with up to 80 pins, DAB 270 supports up to 160 pins, DAB 272 supports up to 240 pins, and DAB 274 supports up to 320 pins.

Before describing the elements of the stand-alone section in detail, the following general statements regarding the HMS are provided to assist in understanding the operation of the elements of the stand-alone section, their interrelated operation, and the operation of the HMS generally.

The Pattern Bus, and therefore, the HMS, may be operated in one of two modes: "access" mode and "presentation" mode. The HMS operates in the access mode whenever the HME is not being evaluated. The evaluations take place during the presentation mode. During the access mode, the CPU may access the PAM, the pattern-block link table of the PAC, the PELs, and various control and status registers of the HMS. The CPU is capable of programming the various components of the HMS, performing error analysis, and reading and writing addresses. During the presentation mode, patterns are sent to the HME being modeled via the Pattern Bus.

During the presentation mode, an HME is evaluated as either a private or a public device. Private devices retain their internal state between evaluations and, therefore, do not have to have their state restored between such evaluations. Private devices can not be shared between device instances or users, because their internal state must be preserved for the next evaluation. Public devices can be shared between device instances or users because their states are restored before each evaluation.

Static devices may be modeled as either private or public devices. Dynamic devices, however, must be modeled as public devices.

CPU

CPU 240 preferably is a conventional microprocessor-based single-board computer based on the MC68020 microprocessor manufactured by Motorola Semiconductor. The detailed design of an equivalent CPU is well within the capability of one skilled in the art. CPU 240 executes all operational code, including boot, diagnostic, and HMS control functions. An on-board 128 KByte erasable programmable read only memory ("EPROM") contains the boot and diagnostic code. An on-board Ethernet® controller (preferably the AM7990 manufactured by Advanced Micro Devices) interfaces with the Ethernet®. The controller runs the lowest levels of network protocols, while the higher levels are run by the CPU. CPU 240 preferably accesses 4 MBytes of dynamic random access memory ("DRAM"). Approximately 1 MByte is used for programs, and approximately 3 MBytes are used for data. The CPU accesses the rest of the stand-alone section of the HMS through TG 248, during the access mode. The CPU control of the PACs, PAMs, and PELs during the access mode will be described in conjunction with the detailed description of each of those elements. CPU 240 directs the PACs to drive programmed pattern sequences over each of the Pattern Bus lanes to the PELs. The HMS supports 80 bits of pin data per lane at up to 25 MHz, therefore, the Pattern Bus throughput can reach a maximum of 8 billion pin-patterns/second during presentation mode.

Pattern Bus

The Pattern Bus at 250, 252, 254, and 256 is operated in the access and presentation modes. The PACs with their attached PAMs and the PELs with their attached DABs connect to the Pattern Bus. The CPU can cause the Pattern Bus to enter the presentation mode. When it does, the PACs drive pre-stored pattern sequences down each lane to the PELs. CPU 240, however, cannot access any of the elements connected to the Pattern Bus, except TG 248, during the presentation mode. TG 248 distributes a number of precision clocks and timing signals on the Pattern Bus. These are identically driven on all four lanes.

Timing Generator

An overall description of the TG and its operation will first be presented. The TG performs two main functions within the HMS. First, it provides a gateway for CPU communication to the PACs, PAMs, and PELs. Second, it controls the presentation of pattern sequences. The TG generates all system clocks and timing strobes as well as supervising the transition between backplane access and presentation modes. During presentation, the TG monitors the lanes involved for synchronization. If the lanes are out of synchronization, the presentation is aborted and an error is reported to the CPU.

The CPU communicates with the TG using a dedicated interface. The TG decodes each access from the CPU into TG, PAC, PAM, or PEL accesses. Accesses to the TG may occur at any time. Since accesses to the PACs, PAMs, and PELs are via the backplane, they can only take place when the backplane is in access mode.

The two modes of the backplane are referred to as access mode and presentation mode. In access mode, the TG repeats the CPU bus signals on the backplane, and vice-versa, allowing communication from the CPU

through the TG to the PACs, PAMs and PELs. Access mode is the default mode and is entered upon power-up or after a system reset.

During presentation mode, PAM contents are clocked out of the PAMs, through the PACs, onto the backplane, and into the PELs and CICs. Using timing strobes generated by the TG, the PEL formats the patterns, as required, and presents them to the HME. During an evaluation of an HME, the backplane always starts in access mode, then transitions to presentation mode, then transitions back to access mode. A state machine on the TG controls these mode transitions. Any attempt by the CPU to access a PAC, PAM, or PEL while the backplane is in presentation mode results in a CPU "bus error."

The TG generates all of the system clocks used during pattern presentation. The system clocks preferably are derived from a phase-locked frequency synthesizer. The reference for this synthesizer is a precision quartz crystal oscillator of preferably 30 MHz.

The TG generates two types of timing strobes for use by the PEL. The first type, referred to as EDGES, is used by the PEL to format the pattern data before presenting to the HME. The EDGES are actually pulses preferably 8 ns wide. A total of six EDGES are generated by the TG. The second type of strobe, referred to as SAMPLE, is used by the PEL to measure the response of the HME to the last pattern in a pattern sequence (the "measurement pattern"). Each stimulus pattern presented to the HME has an associated group of EDGES. Each edge can be programmed to occur at a different time relative to the start of the pattern clock. A single SAMPLE strobe is generated at the end of each pattern sequence.

The TG contains two separate, but similar, ramp generators, one from which the EDGES are derived, and the other from which the SAMPLE strobe is derived. The ramp generators are created by charging a capacitor using a constant-current source and rapidly dumping the capacitor charge at appropriate intervals. The two constant current sources each have four selectable current levels, creating four slopes for each ramp. The ramp voltage thus increases at a software selectable but constant rate. By comparing the ramp waveform to a programmable reference voltage level, time delays can be generated that are proportional to the programmed voltage.

The EDGE ramp is synchronized to the system clock. The duration of the EDGE ramp can be selected to be from one to eight pattern clock cycles. This is the interval in which the six EDGES can be positioned under control of the CMC software running on the CPU. Ramp generation preferably is enabled by control bits from the PACs and PAMs which are driven on the backplane during presentation mode. When the proper code is present on these control bits, a ramp cycle is triggered. At the end of the programmed ramp duration, the ramp is automatically dumped.

Six separate comparators are connected to the EDGE ramp signal. Each comparator connects to an individually programmable reference voltage. Each comparator triggers a pulse generator when the EDGE ramp voltage crosses the reference voltage. In this way, six separate EDGES are generated from the same ramp waveform. The EDGE times are all relative to the trigger point of the EDGE ramp.

A separate comparator with a separate programmable reference voltage is connected to the SAMPLE

ramp signal. When the SAMPLE ramp voltage crosses this reference voltage, it triggers the SAMPLE strobe rising edge. The falling-edge time of the SAMPLE strobe is programmable in units of pattern clock cycles. In this way, the rising edge of SAMPLE strobe is precisely adjustable. The rising edge preferably is used for two-state sampling of the HME pins by the CIC. The falling edge preferably is used for five-state sampling, and does not require precise positioning. It is important that the five-state measurement not be corrupted by noise. Therefore, during the SAMPLE strobe, the TG disables all system clocks, disables the feedback out signal, and instructs the PACs and PAMs to inhibit their memory refresh.

In normal five-state sampling, the HME pins are sampled at the end of a pattern sequence presentation. In this case, the two-state SAMPLE edge is simply ignored. In timing measurement, a pattern sequence is repeated many times, each time moving the two-state SAMPLE edge and determining whether or not the HME output pin has transitioned by that time. By noting the time at which the HME output pins transition and subtracting from it the time at which the causative HME input pin transitions, the actual HME output delays are determined. The same SAMPLE ramp slope is used to detect both the causative input pin transition and the output transition, in order to minimize error.

The reference voltages used for all of the ramp comparators are supplied by a programmable, precision DAC (digital to analog converter). The placement resolution for the EDGES or SAMPLE is dependent upon the resolution available from the DAC and upon the slope of the ramp. Timing measurements are computed from knowledge of two threshold settings of the SAMPLE DAC: one corresponding to the causative input transition, and one corresponding to the resulting output transition.

The accuracy of the EDGE placements and the resulting timing measurements depend on the accuracy to which the ramp slopes are known. The TG contains dedicated hardware that, along with calibration routines in the CMC software, allows measurement of the exact ramp slopes and offsets for each EDGE and SAMPLE. The hardware is a single toggle flip-flop for each EDGE and SAMPLE. The "T" input of this flip-flop is driven from the pattern clock and the clock input is driven from the EDGE or SAMPLE. If the EDGE/SAMPLE rising edge occurs while the system clock is high, the flip-flop toggles, and if it occurs while the system clock is low, the flip-flop does not toggle. After positioning a reference voltage at a particular setting, the time delay corresponding to that setting can be determined by changing the pattern clock period via the programmable phase-locked loop, and then determining whether or not the corresponding flip-flop toggles at that pattern clock period. By adjusting the pattern clock period and repeating the test, a pattern clock period can be found that corresponds to the unknown delay. Since the period of the pattern clock is known precisely, the ramp slope can be calibrated precisely by making two or more measurements.

During pattern presentation, the PAC outputs in the lanes participating in the presentation are compared for agreement of the control bits. If there is any disagreement, the TG latches the offending bits, stops the presentation, and interrupts the CPU.

Now that an overall description of the TG and its operation has been presented, the TG will be described in detail.

Referring to FIG. 7, a block diagram of the TG 248 is shown. CPU interface ("CPU I/F") 302 communicates with CPU 240 on line 301. The signals on line 301 include the control signals from the CPU to the CPU I/F, and the address/data signals that are bi-directionally communicated between the CPU and CPU I/F. The CPU I/F buffers and decodes the signals for use by the remainder of the TG.

The decoded signals are used for Pattern Bus lane and PEL selection, and are used as control signals by the PACs and PELs. Line 317 from the CPU I/F connects the signals for Lane "A" to Lane "A" bus transceiver logic interface ("BTL I/F") 314. In a like manner, line 322 connects the signals for Lane "B" to Lane "B" BTL I/F 324, line 352 connects the signals for Lane "C" to Lane "C" BTL I/F 356, and line 354 connects the signals for Lane "D" to Lane "D" BTL I/F 358.

Pattern control input register 305 receives inputs from CPU I/F 302 on line 304 and from 0-7/control/status register 328 on line 309. 0-7/control/status register 328 is connected to CPU I/F 302 via line 326. The information communicated from CPU I/F 302 to register 328 includes buffered control, status, and data signals. The signals received from the CPU I/F in this case, however, are the valid data signals for each lane. The signals received from the 0-7/control/status register are PEL control signals for each lane.

The output of pattern control input register 305 is input to Pattern Bus error logic 306 via line 313. Pattern Bus error logic 306 includes a programmable array logic ("PAL") which determines whether there are errors in the PEL control data and in the data being placed on any of the lanes.

The output of Pattern Bus error logic 306 on line 308 is input to bus error registers 310, along with the output of pattern control input register 305 on line 313 and the output of 0-7/control/status register 328 on line 330. The signal output from logic 306 is an error signal for the lanes; the signal output from register 305 clears error conditions. If there is an error, the bus error register provides error data to 0-7/control/status register 328 on line 323, and an error signal on line 312 to Lane "A" BTL I/F 314, Lane "B" BTL I/F 324, Lane "C" BTL I/F 356, and Lane "D" BTL I/F 358. The error signal will be processed by the PELs and PACs, and their respective associated circuitry, according to the type of error it is. A detected error will also cause the generation of a CPU interrupt signal that is transmitted back to the CPU by appropriate means.

The output of CPU I/F 302 on line 316 is input to the feedback out select synchronizer/lane play select 318. The outputs from CPU I/F 302 are the buffered lane signals from the CPU. Via bi-directional line 320, feedback out select synchronizer/lane play select 318 receives the feedback signal as an input from the HME, to indicate that a desired HME internal state has been achieved. Feedback out select synchronizer/lane play select 318 processes these signals such that the feedback and the lane select signals are clocked from a register at the same time. The signals are used for conducting pattern play, as will be described. The output of feedback out select synchronizer/lane play select 318 on bi-directional line 320 is input to Lane "A" BTL I/F 314, Lane "B" BTL I/F 324, Lane "C" BTL I/F 356, and Lane "D" BTL I/F 358.

All of the input signals to Lane "A" BTL I/F 314, Lane "B" BTL I/F 324, Lane "C" BTL I/F 356, and Lane "D" BTL I/F 358 have now been described. The outputs of these I/Fs are bi-directionally connected to their respective Pattern Bus lane connectors. That is, the output of Lane "A" BTL I/F 314 is connected to Lane "A" connector 430 by bi-directional line 422, the output of Lane "B" BTL I/F 324 is connected to Lane "B" connector 432 by bi-directional line 423, the output of Lane "C" BTL I/F 356 is connected to Lane "C" connector 434 by bi-directional line 424, and the output of Lane "D" BTL I/F 358 is connected to Lane "D" connector 436 by bi-directional line 426.

The signals output from TTL/ECL (transistor-transistor logic/emitter-coupled logic) converter 334 are used for generating delays for the sample strobes and edge times, and for generating the pattern clock signal. The circuitry for generating the clock signals will be discussed first, then the circuitry relating to the sample strobes and precision edges will be discussed.

The output of TTL/ECL converter 334 on line 336 are the programmed clock rate signals from CPU 240. These signals are input to PLL (phase-locked loop) divide-by-N counter 338. Based on the programmed rate, the signals representative of the clock rate are output on line 340 and input to PLL reference oscillator/phase detector 342. PLL reference oscillator/phase detector 342 compares the frequency of the oscillator-generated signal and the signal output from the PLL divide-by-N counter. The difference in the signals drives a voltage controlled oscillator ("VCO").

The output of PLL reference oscillator/phase detector 342 on line 344 is the reference oscillator signal, which is input to phase-lock detector 346. The second input to phase lock detector 346 is the output of PLL divide-by-N counter 338 on line 340. The two inputs are compared, and when matched, the output of the phase-lock detector on line 348 changes state to indicate to that this has been achieved. The output of phase lock detector 346 is input to 0-7/control/status register 328. This register will place this status information on the bus that communicates with CPU 240.

A second output of PLL reference oscillator/phase detector 342 on line 345 is input to a PLL output divider 347. This output is the output of the VCO, and is used as a clocking signal for PLL output divider 347. The other inputs to the PLL output divider on line 339 from TTL/ECL converter 334 are the signals for determining how to divide the PLL output to obtain the desired clock rate. The input signals are processed, and the output of PLL output divider 347 on line 349 is input to clock synchronizer/selector 350.

Clock synchronizer/selector 350 receives an input from divider 347, and is capable of having two external clocks connected to it. The clock select signals from TTL/ECL converter 334 are also input to clock synchronizer/selector 350, via line 357. After the clock signal is selected, it is synchronized. This clock signal will be used to generate pattern clock signals.

The output of clock synchronizer/selector 350 on line 351 is input to PAC clock driver 410, transmit clock driver 412, and receive clock driver 414. The outputs of the PAC clock driver on line 416, the transmit clock driver on line 418, and the receive clock driver on line 420 are input to each of Lanes "A," "B," "C," and "D" via lane connectors 430, 432, 434, and 436. The PAC clock signal is used for generating the clock signal for the PAC. The receive clock signal is used to direct

the PAC to receive data. The transmit clock signal is used to direct the PAC to transmit data.

Generation of the sample strobes and the edge times will now be discussed. The signals output from CPU I/F 302 on line 360 are input to pattern control input register 362. These signals are the data valid signals for each lane, and the PEL control signals. The output of register 362 is one of the inputs to edge/sample trigger circuitry 368. The other inputs to this circuitry are the control and select signals for the six timing edges from TTL/ECL converter 334 on line 335, and the output of Pattern Bus error logic 306 on line 313.

The outputs of the edge/sample trigger circuitry 368 on line 378 are charging signals for generating the six timing edges, which are input to edge ramp dump buffer 374. The charging signals are input to circuitry that provides analog outputs for the six edges. These analog outputs are input to edge comparators 382 via line 380. The other input to the edge comparators is the output of the edge/sample DAC 376.

The signals input to the DAC are from CPU I/F 302 on line 350. These signals originate at CPU 240, and are converted to analog signals and compared with respective analog signals from edge ramp dump buffer 374. The comparison of the signals at each of the comparators will cause an output on line 384 to 0-5 edge driver 386. The output of the 0-5 edge driver on line 400 is input to each of Lanes "A," "B," "C," and "D," over its respective connector.

With regard to the sample strobes, the inputs to edge/sample trigger circuitry 368 are the sample mode selection signal and clocking signals from TTL/ECL converter 334. The output of edge/sample trigger circuitry 368 on line 373 is input to the ADC (analog to digital counter) sample driver 398. The output signal has a programmed width. The output of driver 398 on line 402 is input to each of Lanes "A," "B," "C," and "D," via its respective connector.

The other output of the edge/sample trigger circuitry, relating to the sample strobe, is on line 372, and it is input to sample ramp dump buffer 378. This output includes charging signals which are processed by sample ramp dump buffer 378 to generate an analog signal. The analog signal is output on line 392 and is input to sample comparator 394, where it is compared with an analog output from the edge/sample DAC. The output of sample ramp dump buffer 378 preferably is also provided as a test point.

The output of sample comparator 394 on line 396 is input to edge/sample trigger circuitry 368. This signal is used to set the register that provides the output signal to ADC sample driver 398 via line 373.

Having described the circuitry of TG 248, aspects of its operation and interrelationship with other elements of the HMS will now be discussed.

As stated, one of the primary functions of TG 248 is to provide a single source of timing signals for use by the HMS. However, TG 248 also performs other important functions. During the access mode, TG 248 provides a path between CPU 240 and the four lanes of the Pattern Bus, permitting the CPU to access the PACs, PAMs, and PELs to perform read and write operations. This is done in the form of read and write requests. The specific requests that can be made through TG 248 will be discussed in conjunction with the elements to which the requests are made.

The pattern clock that is output from TG 248 clocks the patterns from the pattern memory onto the Pattern

Bus. The pattern clock rate is programmable by the CPU and is set according to the "device-speed" parameter specified in the shell software.

The six edge strobes generated by TG 248 are used for formatting HME data and clock input signals. The CMC automatically computes the edge times based on the "device_speed," "device_setup_time," and "device_hold_time" specified in the shell software. TG 248 uses the programmable-slope ramp generator and the six comparators to place the edges with a resolution of approximately 0.5 ns.

The sample strobe is used by the PELs to sample HME outputs. The rising edge of the signal can be swept by the CMC in four ranges, with resolution as fine as 0.5 ns, in order to precisely determine the time of output and I/O pin transitions. The falling edge, which is automatically placed by the CMC, samples the five-state output and I/O pin values.

As briefly discussed above, TG 248 performs error handling functions. The primary function is to detect errors on the four lanes of the Pattern Bus. If an error is detected, an interrupt signal is sent to the CPU. Depending on the type of error, the HMS may discontinue the current evaluation.

Errors detected by TG 248 include parity error, shorted pin error, and DAB error. Parity error detection ensures that proper transmission of patterns takes place and ensures that the patterns are properly received by the CICs on the PELs. A parity error is a fatal error.

The shorted pin error indicates that the CICs of the PEL have detected that a pin driver is shorted. This is not a fatal error.

The DAB error indicates that a DAB has either been inserted or removed, or that the DAB is not active. This is not a fatal error.

TG 248 generates other error signals, in addition to the main signals discussed above. These signals will be discussed in the course of describing the operation of the HMS and the elements that generate the various signals.

The six timing edges that are generated are used for HME data and clock formatting during pattern play. The slope of the analog voltage ramp used to generate the edges is selected to allow all six timing edges to occur within one pattern clock period. The ramp used preferably has a software-selectable slope of 0.5, 2, 10, or 50 ns/threshold.

The time delays of the six timing edges are programmable via an 8 bit address. The 8 bit address selects one of 256 points along the slope. For example, if the 0.5 ns/threshold slope is selected, the minimum timing delay available is: $(0.5 \text{ ns/threshold}) \times (1 \text{ threshold}) = 0.5 \text{ ns}$. The maximum delay is: $(0.5 \text{ ns/threshold}) \times (256 \text{ thresholds}) = 128 \text{ ns}$.

The TG performs timing checks on patterns transmitted by the PACs. It determines whether the pattern sequences transmitted by adjacent PACs stop at the same time, and verifies that patterns are sent to the same PEL in each lane.

TG 248 may also include an on-board clock for diagnostic purposes.

Pattern Controller

Referring to FIG. 8, a block diagram of representative PAC 258 and PAM 259 is shown. The following description of PAC 258 and PAM 514 is representative of the other PACs and PAMs of the HMS. PAC 258

includes timing/CPU interface ("TMG/CPU I/F") 502, memory control 504, link table 506, pattern control 508, refresh control 510, and backplane I/F 512. The pattern memories of the PAC/PAM combination are shown at 514. PAM 514 includes from one to four PAM boards 516, which are connected to a PAC.

An overall description of the PAC and PAM operation will first be presented. The PAC, together with one or more PAMs, allows for the storage and presentation of pattern sequences in the HMS. The HMS preferably can support up to four PACs, one per lane. A minimum of one PAC is required for basic system operation. An edge connector on the PAC allows the PAC to be connected directly to a single lane of the backplane. Because the PAC itself has no memory for pattern storage, stacking connectors on the PAC allow from one to four PAMs to be connected to the PAC. This allows the user to configure and upgrade pattern memory within the system.

Pattern memory is allocated in blocks of 256 patterns by CMC software running on the CPU. A pattern sequence is initially allocated as a single block. Additional blocks of memory are allocated as the pattern sequence length increases beyond 256 during simulation or test-vector play. It is not necessary for the pattern blocks that make up a pattern sequence to be contiguous within the pattern memory, because pattern blocks are linked together by a 32K-by-15-bit "link table." The link table, located on the PAC, contains the information necessary to generate a continuous pattern sequence from the blocks which may be scattered throughout the pattern memory.

The CMC software running on the CPU allocates and deallocates pattern memory in 256-pattern blocks, and maintains pattern sequences as lists linked via the link table. The HMS hardware follows the links in the link table when presenting pattern sequences to the HME. Because of these linked-list pattern-memory allocation and presentation methods, multiple pattern sequences can share pattern memory and can grow and shrink without requiring the CPU to copy patterns from one location in pattern memory to another.

Linked pattern memory blocks are also used to support concurrent fault simulation. In concurrent fault simulation, multiple history sequences are retained in pattern memory. These history sequences represent multiple instances of a single device for different faults of the system being simulated. These multiple history sequences may have initial portions which are common. The HMS stores the common initial subsequence of these differing history sequences only once. It links the common initial subsequence to any of several different ending subsequences by storing the appropriate link in the link table immediately prior to presentation.

The backplane can be placed in either of two operational modes: access mode or presentation mode. The behavior of the PAC is dependent on the mode of the backplane. Between device evaluations, the entire backplane is in access mode. This mode of operation allows the CPU board to access the contents of the status and control registers on the PAC, the link table, and the memories residing on all PAMs connected to the PAC.

During evaluation of an HME, the backplane is in presentation mode. In presentation mode, all PACs operate synchronously, simultaneously presenting patterns to the PELs in each of the enabled lanes. Preferably, each PAC presents 80 data bits, 5 parity bits, and 11 control bits to the backplane during each pattern cycle.

Additional bits from the pattern word are used internally in the PAC.

The patterns presented to the backplane by the PAC during presentation mode are stored within memories on the PAM or PAMs connected to the PAC. Preferably, the pattern words stored within a PAM are 102 bits wide, and consist of 80 data bits, 16 control bits, and 6 parity bits.

Lengthy simulations may require that long pattern sequences be stored in the PAM. The memories used on the PAMs are, therefore, preferably able to output a long, continuous sequence of patterns at a high pattern rate (25 MHz). For this reason, the PAM uses Video RAMs ("VRAMs") for pattern storage.

PAMs preferably are of several different capacities. A 128K PAM preferably uses 256K-bit VRAMs such as the M5M4C264L-12 manufactured by Mitsubishi, and a 512K PAM preferably uses 1M-bit VRAMs, such as the TC524256AZ-12 manufactured by Toshiba. The 256K-bit VRAMs consist of a 64K-by-4-bit dynamic RAM port and a 256-by-4-bit serial read/write port. The serial port allows the data stored within the memory to be clocked out with a high speed clock. The 1M-bit VRAMs are similar, with the dynamic RAM port organized as 256K-by-4 bits and the serial port organized as 512-by-4 bits. During access mode the patterns are stored in the VRAMs using standard DRAM RAS/CAS protocol. During presentation mode, the serial port is used to clock the data out of the memory and onto the backplane via the PAC.

Due to the high pattern rates encountered during presentation, the pattern memories preferably are interleaved. The pattern clock received by the PAC is divided by two, and half of the VRAMs use one phase of the divided clock to shift data out of their serial ports, while the remaining VRAMs use the other phase of the divided clock. Since the VRAMs have a 4-bit-wide serial output, a total of 24 VRAMs are required to store 96 bits of pattern and control data. The six additional parity bits require two more memory devices, for a total of 26 VRAMs per pattern word. Because the memories are interleaved, twice this number (52 VRAMs) preferably reside on the PAM.

During access mode, pattern memory is divided into three banks in order to make efficient use of the memory address space allocated to the PAC and connected PAMs. The 96-bit pattern/control word is split into three 32-bit words, each accessed in a different memory bank. Each 32-bit bank is further divided into a 16-bit low data word and a 16-bit high data word, allowing for 16-bit accesses from the CPU. The parity VRAMs are duplicated because of interleaving, that is, there are a total of four parity VRAMs on the PAM, two for each interleaved half of pattern memory. Only three of the four bits are used on each VRAM, one for each of the three memory banks.

During CPU write accesses to the PAC/PAM, two parity circuits on the PAC generate separate parity bits for the two 16-bit words. These parity bits are sent to the PAMs where they are written into the appropriate parity VRAMs. One parity bit is individually written into each affected parity VRAM by making use of the "write-per-bit" feature of the VRAMs. During a 32-bit write access, two parity bits are generated and written simultaneously. During a 16-bit write access, only one of the two parity bits is valid, and only that bit is written into the VRAM. Each parity generator preferably can be configured to output either even or odd parity.

When the CPU performs a read access to pattern memory, the parity bit or bits associated with the accessed location are read from the appropriate parity VRAM and routed to the parity circuitry on the PAC.

5 The remainder of the pattern word is also sent to the parity circuitry, which computes the parity and compares the result with the parity bits. If the result indicates a parity error, then the CLK_READ signal latches the pattern word and the address at which it occurred. During a 16-bit access, only the 16 bits being accessed can generate a parity error.

In addition to the two parity generator/checkers which operate during access mode, there is another 16-bit parity checker which operates during presentation mode. This parity checker computes parity on the 16-bit pattern control word. The generated parity bit is compared with the appropriate parity bit stored in pattern memory. If a parity error is detected, the PAC halts pattern presentation.

20 The five parity bits associated with the 80 bits of pattern data are driven onto the backplane during presentation mode. The PEL receives these five parity bits, and the CICs on the PEL use them to check the parity of each 16-bit section of the 80-bit pattern data. If a parity error is detected by any CIC on the PEL, the PEL asserts the backplane ERROR* signal. The PAC receives the ERROR* signal and immediately halts pattern presentation.

During a CPU access, the low 32 pattern bits on the 30 backplane are used by the PAC as address bits. When the backplane address strobe is asserted, the lane select and address bits are latched by the PAC. Decode logic on the PAC uses the backplane address strobe, the latched lane select, and address bits 20 through 27 to determine if the CPU is addressing the PAC/PAM. If it is, the PAC generates an internal signal indicating that a CPU request is in progress. A request code is sent to the memory control circuitry to determine what type of transaction the control circuitry should perform.

40 All memory and register requests are processed by the "request machine" circuitry of the PAC. This state machine is composed of two PROMs with registered outputs, and uses a 16 MHz crystal oscillator as its clock. All requests except pattern memory refresh requests are asynchronous to the 16 MHz clock and are first synchronized with a two stage flip-flop synchronizer.

Because stacking connectors preferably are used to connect multiple PAMs to a single PAC, each PAM is strapped with a two bit address. This is accomplished using a DIP switch on each PAM. Each DIP switch is set to a unique address, starting with zero. If both 512K and 128K PAMs are installed on the same PAC, the 512K PAMs are assigned the lowest switch settings. In order to verify that the switches have been set properly, the PAC and PAM contain memory detection circuitry. An analog comparator circuit on the PAC automatically determines the number of PAMs plugged into the PAC. This number can be read by the CPU.

60 When the CMC, running on the CPU, writes data patterns in the PAM, it also stores a 16-bit control word with each of the patterns. Bits D0 through D3 of the control word are PEL address bits. These bits are driven onto the HMS backplane during pattern presentation and are used to select the appropriate PEL within the lane. Bits D4 through D6 are PEL control bits. These bits are driven onto the HMS backplane during pattern presentation and are used by both the TG and

the PELs. Bit D7 is the USER bit. It is driven onto the HMS backplane during pattern presentation and is received by the PEL. The PEL then drives the USER bit onto a test point on the DAB. Bits D11 through D13 are spare control bits which are unused by the PAC. Bits D14 and D15 are spare bits driven onto the backplane during presentation.

Bits D9 and D10 of the control word are branch command bits. These bits, in conjunction with the link table, instruct the PAC to stop fetching patterns from the current block and to begin fetching in the next block. The branch command code tells the PAC when to jump to the next block, and the link table contains the address of the next block. The transfer from one block to the next is accomplished by performing a "serial read transfer cycle" operation within the VRAM, transferring a new row of data from the dynamic RAM port to the serial shift register port.

Having presented an overall description of the PAC and PAM operation, a detailed description of the PAC will now be presented.

In order to understand operation of the PAC in both operational modes, it is necessary to understand the signals at backplane I/F 512. The Pattern Bus preferably carries signals of two logic families: BTL (as supported by BTL transceivers manufactured by National Semiconductor) and ECL. This was described in discussing the signals placed on the Pattern Bus by TG 248. BTL transceivers are used for the majority of the Pattern Bus signals. The ECL drivers and receivers are used for the more critical timing signals such as clocks, timing edges, and strobes. Table 2 shows the signals at the backplane I/F and their descriptions. Many of these signals have different functions for the access and presentation modes; this is reflected in the descriptions.

TABLE 2

Signal	Description
RESET	The RESET signal is an active high BTL input signal driven by the TG. This signal is asserted upon power-on, or when the HME is initialized. While the RESET signal is asserted, refresh cycles cannot take place, and the contents of the pattern memory may be lost.
PLAY*	The PLAY* signal is an active low input signal driven by the TG. The state of the PLAY* signal determines the state of the Pattern Bus lane. PLAY* is inactive (high) during access mode, and is asserted during presentation mode.
ERROR*	The ERROR* signal is an active low BTL "party-line" signal driven by the TG, the PEL, and the PAC itself. During access mode, the PAC ignores ERROR*. During presentation mode, the PAC halts pattern play whenever ERROR* is asserted. The PAC asserts the ERROR* signal when an internal error condition is detected, independent of the Pattern Bus mode. The ERROR* signal is asserted until the error condition is cleared by the software or until the RESET signal is asserted.
PARITY<0..4>*	The five Pattern Bus parity signals, PARITY<0..4>*, are BTL signals which serve different purposes during each operational mode. During the pattern presentation mode, the signals are the parity

TABLE 2-continued

Signal	Description
	signals for the 80 pattern bits, one parity bit per sixteen pattern bits. The parity bits are stored in pattern memory and are driven onto the Pattern Bus by the PAC. During CPU access mode, the parity signals serve as control signals driven by the TG and received by the PAC.
	In the access mode: PARITY<0>* is the AS (address strobe) signal. This AS signal is an active high signal. This signal indicates when the address line signals are valid.
	PARITY<1>* is the READ (read data) signal. The READ signal determines the direction of CPU transfers. The READ signal is high when the CPU is reading data from the PAC or PAM. The READ signal is low when the CPU writes to the Pattern Bus.
	PARITY<2>* is the DS (data strobe) signal. The DS signal is an active high signal and indicates the data portion of a CPU transfer cycle. During CPU write cycles, the assertion of the DS signal indicates that valid data exists on the shared address/data bus. During CPU read cycles, the assertion of the signal indicates to the device being accessed that it may place data on the Pattern Bus.
	PARITY<3>* is the LW (long word) signal. This active high signal determines the width of data transfers. When LW is low, the CPU cycle is a 16-bit transfer. When LW is high, a 32-bit transfer is performed.
	PARITY<4>* is the LANE (lane select) signal. The active high LANE signal is generated by the TG from CPU address bits. The PAC and PEL have no way of sensing which backplane lane they are in. Hence, the LANE signal acts as a lane select.
FB_IN*	The FB_IN* signal is an active low BTL "party-line" signal driven by the PAC and PEL. FB_IN* serves as an acknowledge signal asserted by the PAC at the end of the CPU access mode. During pattern presentation mode, the FB_IN* signal is used to transmit the feedback signal from the PEL to the TG.
FB_OUT*	The PAC simply drives FB_IN* with a logic "1" value during this time. The FB_OUT* signal is a BTL signal driven by the TG. It is received by the PAC and is used as the HME feedback signal during pattern presentation mode. During CPU access mode, FB_OUT* signal is ignored by the PAC.
PAC_CLK	PAC_CLK signal is a differential ECL signal that is used by the PAC and the PAM for clocking the majority of the control and data signals.
XMT_CLK	XMT_CLK signal is a differential ECL signal that is used to register the pattern bits output onto the Pattern Bus during pattern presentation.
RCV_CLK	RCV_CLK signal is a differential ECL signal that is used to clock the PAC inputs during presentation

TABLE 2-continued

Signal	Description
SAMPLE	mode. The SAMPLE signal, is a differential ECL signal driven by the TG. Whenever the received SAMPLE signal goes high, the PAC completes any refresh cycle it is in and withholds refresh cycles until the SAMPLE signal returns to the low state.
PEL_CTL<0...2>*	The three PEL control bits, PEL_CTL<0...2>*, are stored in the PAM as part of the pattern control word and are driven onto the Pattern Bus by the PAC during the pattern presentation mode. During CPU access mode, the PAC receives, but ignores, the PEL control signal, in order for the TG to drive them onto the Pattern Bus.
DATA_VALID*	The DATA_VALID* signal is a BTL signal that is driven by the PAC. This signal indicates when the Pattern Bus control and pattern bits are valid. Several clock cycles after presentation mode is initiated by the TG, the DATA_VALID* signal is asserted, indicating that the next clock cycle contains valid data and control signals. When the DATA_VALID* signal goes high, data and control signals are valid for only one more clock cycle.
PEL_ADDR<0...3>*	PEL_ADDR<0...3>* are the PEL address signals. These signals are BTL signals which are driven by the PAC. These signals, stored in the PAM as part of the pattern control word, are only valid during the pattern presentation mode.
SPARE CONTROL	Spare control signals may be provided. These signals are similar to the PEL address bits in that they are driven by the PAC, but are only valid when the DATA_VALID* signal is active. The spare bits are stored in the PAM. In the preferred embodiment, the spare signals are not used by the TG or PEL.
USER*	The USER* signal is a BTL signal. The USER* signal is stored in the PAM as part of the pattern control word. It is only valid during the pattern presentation mode.
PAT<0...79>*	These signals are BTL signals. During pattern presentation mode, the 80 signals contain pattern data and are driven by the PAC onto the Pattern Bus. During CPU access mode, 48 of the signals, PAT<32...79>*, are not driven. The remaining 32 pattern signals, PAT<0...31>*, are used as multiplexed address/data signals. They are driven by the TG during the address portion of the cycle and during the data portion of CPU write cycles. The PAC drives them during the data portion of CPU read cycles.

FIG. 9 is a schematic diagram of TMG/CPU I/F 502. The TMG/CPU I/F uses the control strobes, the PLAY* signal (which is inverted to become the PRESENT signal), and the lower 32 bits of the Pattern Bus to generate separate address and data buses, as well as

various control signals for use by the PAC and the PAM, as will be described.

Line 525 represents the lower 32 bits of the Pattern Bus, which may be within any one of the four lanes of the Pattern Bus. The lower 32 bits on line 525 are input to register 528. The READ_DATA* signal on line 546 is input to tri-state buffers 532 and 536. The READ_DATA* signal determines the direction of data flow through the Pattern Bus. When the READ_DATA* signal is a logic high value, the direction of data flow is from line 525 to line 542; when it is a logic low value, data flow is in the opposite direction. In the opposite direction, data on the PAC address/data bus is input to register 526 and then placed on Pattern Bus 525. The CLK_READ signal on the 559 clocks shift register 526.

The output of buffer 536 on line 542, which is the PAC address/data bus, is input to transceiver 538 and latch 540. Transceiver 538 provides bi-directional communications between PAM data bus 548 and the PAC address/data bus 542.

The second input to latch 540 is the lane enable signal on line 530. The outputs are the signals on address buses 552 and 556. The address bus signals are input to the decode logic 520 and sent to other portions of the PAC. The decode logic comprises two PALs. The other inputs to the decode logic are the AS* signal on line 558, the DS* signal on line 560, and the READ* signal on line 561. Decode logic 520 processes the input signals, and provides at its outputs the READ_DATA* signal on line 546, the CPU_REQ signal on line 534, the REQ_CODE on line 535, and various memory and register strobes at line 538. The READ_DATA* signal was explained above. The CPU_REQ signal, the REQ_CODE signal, and various memory and register strobes will be described in discussing the PAM and the remainder of the PAC.

During the access mode, the PLAY* signal, driven by TG 248, is deasserted on all four lanes of the Pattern Bus. TG 248 drives the address strobes in the deasserted state until a CPU read or write cycle takes place. While the address strobe is in its deasserted state, the lane enable signals and the lower 32 bits of the Pattern Bus are received by the PAC through latch 540.

At the beginning of a CPU access, the lower 32 bits of the Pattern Bus become address bits and, if the access is directed at the lane, the lane enable signal becomes active. When the address strobe becomes active, the lane enable and address bits are latched. Decode logic 520 uses the address strobe, the latched lane enable, and address bits 20-27 to determine whether the CPU is addressing the PAC/PAM. If it is, a CPU request signal is generated. Decode logic 520 also uses other control signals such as the RD* and DS* signals to generate various control strobes and memory selects.

The three bit REQ_CODE signal 535 is sent to the PAMs and determines the type of transaction the control circuitry must perform. The REQ_CODE signal is encoded as shown in Table 3. The first two entries in the table are for transfer cycles which occur only during the presentation mode. The remaining codes are for PAM reads and writes, and link table or register reads and writes.

TABLE 3

Bits			Transaction
R1	R2	R3	
0	0	0	ASYNCFER

TABLE 3-continued

Bits			Transaction
R1	R2	R3	
0	0	1	SYNC XFER
0	1	0	ILLEGAL
0	1	1	ILLEGAL
1	0	0	DRAM WRITE
1	0	1	DRAM READ
1	1	0	LINK/REG WRITE
1	1	1	LINK/REG READ

The use of each transaction will be described in conjunction with the description of link table 506 and the PAM.

FIG. 10 is a schematic drawing of memory control circuitry 504. Memory control circuitry 504 processes all PAM, link table, and register requests. All of these requests, except the REF_REQ* signal on line 611, are multiplexed on line 613 as the UNSYNC_REQ* signal. The specific requests that are input to the memory control circuitry will be discussed after the disclosure regarding the circuit and its operation.

Memory control circuitry 504 includes programmable read only memories ("PROMs") 614 and 616, and flip-flops 602, 604, 606, 622, and 647. Flip flops 602, 604, and 606 are D-type flip-flops that are used to synchronize the UNSYNC_REQ* signal. The UNSYNC_REQ* signal, as stated, includes all of the request signals except the REF_REQ* signal on line 611. The three flip-flops are clocked by the 16 MHz XTAL_CLK signal on line 720 that is output from the crystal clock on the PAC. Following conventional clocking of the three flip-flops, the SYNC_REQ* signal is output from the "Q" output of flip-flop 606 on line 612. This signal is input to PROMs 614 and 616. The CLK_REQ signal is output on line 607 from the "Q" output of flip-flop 606 to clock flip-flop 622.

The remaining inputs to PROMs 614 and 616 will now be described. The REF_REQ* signal on line 738, which is output from refresh PAL 714 (FIG. 11), is input directly to PROMs 614 and 616. The next input to the PROMs is the output of flip-flop 622. The inputs to the flip-flop are the PLAY_CMD* signal on line 618 and the three bit REQ_CODE signal on line 536.

The last input to PROMs 614 and 616 is the 4-bit REQ_STATE signal on line 635. This signal is output from PROM 614 and then input to both PROMs. The REQ_STATE signal inputs to the PROMs the current request type being processed by the PROMs. Outputs of PROM 614 include the synchronous control signals that are sent to the PAM after being gated with other signals. These signals are the ASYNC_RAS* signal on line 626, ASYNC_CAS* signal on line 628, ASYNC_C_TR/OE* signal on line 630, and ASYNC_COL* signal on line 632.

The first output of PROM 616 is the CPU* signal on line 633. This signal and the PAM* signal on line 631 are input to OR gate 641. The PAM* signal is an output of decode logic 520 of the TMG/CPU I/F 502. The output of gate 641 is the PAM_CPU* signal on line 634 which is sent to the PAM Decode PAL 1050 (FIG. 19).

The next three outputs of PROM 616 are control signals that are sent to the PAM. These signals are the WS* signal on line 636, the XFER* signal on line 638, and the REF* signal on line 640. The WS* signal controls PAM write operations, and the XFER* and REF* signals are input to PAM RAS generation PALs 1064 and 1066 (FIG. 19) for generating various RAS signals.

The XFER* signal is for effecting transfers from memory to a shift register. The REF* signal is the refresh enable signal.

The REF_ACK* signal on line 642 is the next output of PROM 616. This signal is input to the Refresh PAL 714. The REQ_ERR* signal on line 644 is output from PROM 616. The REQ_ERR* signal, when asserted, is used to generate the ERROR* signal which is placed on the Pattern Bus. The CLK_READ signal on line 559 clocks shift register 526 as described, and clocks the flip-flops for use in determining parity error.

The last output of PROM 616 is the CLK_ACK signal on line 656. This signal is input to the clock input of flip-flop 647. The input to the clear input is the CLR_ACK* signal on line 657. This signal is the AS* signal on line 558 after being inverted by inverter 655.

Memory control circuitry 504 processes link table and register read requests in exactly the same way since they are multiplexed as the UNSYNC_REQ* signal on line 613. A request is processed in two clock cycles. During both cycles, the COL* strobe and the signal on line 632 are asserted. Although these signals are asserted during all register and link table read operations, they are only used during reading of the PROM of the PAM. The CAS signal enables the lower address bits onto the row-column address bus. This bus connects to the output of the PAM ID PROM. The CPU signal, when gated with the PAM signal, turns the PAM data bus drivers on, allowing PAM ID PROM data to be read. Also, during the first of the two cycles, the CLK_ACK signal on line 656 is asserted. The rising edge of this active high signal clocks flip-flop 647, which asserts the ACK signal on the Pattern Bus. Even though valid data does not yet exist on the Pattern Bus, the negative setup time from data to acknowledge required by the TMG/CPU combination allows the PAC to assert the ACK signal two clock cycles before data is available.

During the second cycle, the CLK_ACK signal is deasserted and the CLK_READ is asserted. The rising edge of this active low signal clocks the desired data into the shift register of the TMG/CPU I/F section.

Memory control circuitry 504 then returns to its default state, deasserting the CAS signal, the CPU signal, and the CLK_READ signal. The backplane Pattern Bus ACK signal is automatically deasserted when the AS* is deasserted by the CPU, clearing flip-flop 647.

Link table and register write requests, like read requests, are processed in two clock cycles. During both cycles, the WS* signal is asserted. This active low signal is gated by various decoded memory/register select signals to generate the appropriate address select.

PAM read requests are processed in five clock cycles. During cycle 1, the CPU* signal on line 633 and the RAS signal sent to the PAM are asserted. These signals remain asserted during all five cycles. During cycle 2, the EVEN_TR/OR or ODD_TR/OR signal, and the COL* signal are asserted. In cycle 3, the CAS signal is asserted. In cycle 4, the CLK_ACK signal (and therefore the Pattern Bus ACK signal) is asserted. During cycle 5, the CLK_ACK signal on line 656 is deasserted and the CLK_READ clock is asserted. After cycle 5, all signals which remain asserted are deasserted. The Pattern Bus ACK signal is deasserted when the AS* signal is deasserted by the CPU, clearing flip-flop 647.

PAM write requests are also processed in five clock cycles. During cycle 1, the CPU signal is asserted. During cycle 2, the RAS and the CLK_ACK signals are asserted. In cycle 3, the COL* signal and the WS* are

asserted and the CLK_ACK is deasserted. In cycle 4, the CAS signal is asserted. In cycle 5, all signals remain as they are. After cycle 5, all signals which remained asserted are deasserted. The Pattern Bus ACK signal acknowledge is deasserted when the AS* is deasserted by the CPU, clearing flip-flop 647.

Shift register transfer requests occur during pattern presentation. Each of the video memories ("VRAMs") of the PAM contains a serial shift register. This register is as long as one of the rows within the DRAM array (256 cells for VRAMs on a 128K PAM). Any row within the VRAM may be transferred to the shift register by performing a shift register transfer cycle. The contents of the shift register may be clocked out of the memory serially. The PCLK signal, divided by two, is used as the serial clock.

The row address selects the row to be transferred to the register. When the transfer takes place, the column address selects one of the registers as the first to be clocked out; this is the start address. If the register contents have been clocked out and another transfer does not take place, the same data from the start address is output again. That is, the shift register is cyclic.

At high PCLK rates, the placement of control signal transitions (RAS, CAS, etc.) relative to the serial clock is critical. These signals, therefore, must be generated by the PCLK signal itself. Pattern control 508, which uses the PCLK clock, generates the transfer control signals. This type of shift register transfer is a synchronous transfer, because the control signals are synchronous with PCLK.

Synchronous transfers are used at PCLK frequencies greater than 1 MHz, while asynchronous transfers are used below this frequency. A PCLK signal flip-flop on the PAC is set or cleared by software, depending on the frequency of the next pattern play. The output of this flip-flop is used by decode logic 520 when forming the three bit REQ_CODE signal. The state of this bit determines whether the request is a synchronous or an asynchronous transfer request.

When pattern control 508 sends a synchronous transfer request, memory control circuitry 504 finishes any refresh cycle which may be in progress, and then asserts the XFER* signal on line 638. This active low signal is buffered and sent to the PAMs where it is used as a RAS enable signal. The memory control circuitry asserts the XFER* signal until pattern control has completed the transfer operation and has removed the transfer request.

When pattern control 508 sends a synchronous transfer request, the request state machine finishes any refresh cycle which may be in progress, and then begins the six cycle transfer sequence. During cycle 1, the XFER* signal and the EVEN_TR/OE or ODD_TR/OE enable signals are asserted. In cycle 2, the RAS signal is asserted. In cycle 3, the COL* signal is asserted. During cycle 4, the CAS signal is asserted. In cycle 5, all signals remain asserted. After cycle 6, all signals are deasserted and the transfer is complete.

Refresh requests are not sent through the synchronizer because they are generated by the refresh control circuitry 510, which uses the same 16 MHz clock as memory control circuitry 504. Refresh requests, therefore, are sent directly to the memory control circuitry. They are processed only when the memory control circuitry is idle. Each CAS-before-RAS refresh sequence is four cycles long. During cycle 1, the REF* and the CAS signals are asserted. During cycle 2, the

RAS signal is asserted. In cycle 3, the CAS signal is deasserted and the REF_ACK* signal is sent back to refresh control circuitry 510, to indicate that the REF_REQ* signal is being processed. During cycle 4, the REF_ACK* and the RAS signal are deasserted. After this cycle, the REF* signal is deasserted.

To minimize the size of the current pulse during refresh, only one PAM board is refreshed at a time.

Referring now to FIG. 11, to achieve a REF_CLK* signal with the right period for the refresh sequences, the 16 MHz clock is divided by 62 using 4-bit counters 704 and 706. This provides a REF_CLK* signal with a period of about 3.87 ms, which is within and approximates the maximum refresh request period of 3.9 ms.

The REF_CLK* signal resets counters 704 and 706, and is asserted at the input of flip-flop 708. Flip-flop 708 is clocked by the XTAL_CLK signal on line 718. Exclusive-OR gate 710 processes the REF_ACK* and REF_CLK signals and outputs the EN_REF_CNTR signal on line 726 to the parallel enables of counter 712. The output of counter 712 is the REF_RCO* signal on line 732, which is input to refresh PAL 714. Since transfer cycles cannot be interrupted, the refresh circuitry preferably can accumulate REF_CLK signals. Accordingly, a burst of refresh cycles are performed immediately after the transfer cycle is complete.

Refresh cycles are accumulated during the HME sampling time and during normal CPU accesses. Refresh PAL 714 provides at its output a 2-bit address for input to the PAM for performing refresh cycles. The second output is the REF_REQ* signal on line 738. The other output is the MUST_REFRESH* signal on line 740 and the REF_ERR* signal on line 734.

During synchronous transfer cycles, refresh control circuitry 510 may accumulate up to seven REF_CLK signals. This occurs at the minimum pattern clock rate used for synchronous transfers of 1 MHz. During CPU accesses and asynchronous transfer cycles, only one or two refresh clocks usually need to be accumulated. Holding refresh cycles while the sample strobe is active also causes REF_CLKs to accumulate. The width of the sample strobe determines the number of refresh requests accumulated.

The MUST_REFRESH* signal is also input to flip-flop 716. This flip-flop is clocked by the STOP_REFRESH* 3 signal. The output of the flip-flop (from the "Q*" output terminal) is the L_MUST_REFRESH* signal on line 735. The STOP_REFRESH will be used to clock the flip-flop when a pattern presentation starts or when the MUST_REFRESH* signal is asserted.

FIGS. 12 and 13 show pattern control circuitry 508. This circuitry includes, for the most part, a series of interconnected PALs and a PROM. Pattern control circuitry 508 uses the PCLK signal and, therefore, is only active during pattern presentation mode. Pattern control 508 performs several functions. It determines when the Pattern Bus changes from CPU access mode to pattern presentation mode, detects error conditions which occur during pattern presentation, indicates that a pattern-block branch must take place, indicates that a transfer operation is in progress, and indicates when a transfer cycle is complete.

Pattern Control 508 includes transfer PAL 740, Pattern control PAL 742, buffer control PAL 744, request PAL 746, transfer PROM 748, 4-bit counter 750, and register 752. Transfer PAL 740 includes pattern transfer circuitry and the branch circuitry.

The pattern transfer circuitry has one internal and three external inputs and a single output. The external inputs are the PAC_PLAY signal on line 762, PAC_ERROR signal on line 763, and XFER_FB* signal on line 764. The PAC_PLAY signal is the PLAY signal that has been buffered, inverted, and registered. A change in state of this signal informs the pattern transfer circuitry when the operating mode changes between the access mode and presentation mode. The PAC_ERROR signal is the ERROR signal on line 763 after it has been buffered, inverted, and registered. A change in state of this signal informs the pattern transfer circuitry that an error has been detected during pattern presentation.

The BRANCH signal is an output of the branch circuitry and is internally connected to the pattern transfer circuitry. This signal indicates that a pattern-block branch must take place. The XFER_FB* signal is an output of transfer PROM 748 and registered by register 752. It indicates that a transfer cycle is complete. The output of the pattern transfer circuitry is the PAT_XFER* signal on line 756. This signal is input to pattern control PAL 742 and request PAL 746 via lines 754 and 756, respectively. When asserted, this output indicates that a transfer operation is in progress.

The branch circuitry of transfer PAL 740 has three inputs. These are the 2-bit B_CMD signal on line 765 and the FEEDBACK signal on line 766. The output of the circuitry is the previously discussed BRANCH signal. The B_CMD bits reside in the PAM and are clocked into the PAC during pattern play. Prior to the 2-bit B_CMD signal being input to the transfer PAL, it is gated with the CNTL_VALID signal on line 784, which output from buffer control PAL 744. Therefore, when the pattern data is not valid, the signal will not be asserted. The FEEDBACK signal is the registered FB_OUT* signal. The output of the branch circuitry is the BRANCH signal, which is sent to the pattern transfer circuitry via an internal connection.

As stated, the PAT_XFER* signal on line 754 is input to pattern control PAL 742. The other inputs to the PAL are the RESET* signal on line 767, the PAC_PLAY signal on line 762, the PAC_ERROR signal on line 763, the XFER_FB* signal on line 764, and the PAC_STOP signal on line 768. The PAC_PLAY, PAC_ERROR, and XFER_FB* signals have been previously described, and those descriptions apply here. The RESET* signal is the Pattern Bus RESET* signal that has been buffered. The PAC_STOP signal is similar to the 2-bit B_CMD signal sent to transfer PAL 740. The PAC_STOP signal is asserted in case the PAC_ERR* signal is asserted while a transfer is taking place. This will allow the transfer to be completed.

The pattern control PAL has five outputs. These are the VALID signal on line 769, the ODD_OUT* signal on line 775, the MUX_EN* signal on line 776, and the BUF_EN* signal on line 780. The MUX_EN* signal is a PAM enable signal that is input to buffer 777. The ODD_OUT* signal is also input to buffer 777. The output of buffer 777 is the PAM_ODD_OUT* signal on line 778 and the PAM_MUX_EN* signal on line 779. When the PAM_MUX_EN* is asserted, it enables the pattern data multiplexers on the selected PAM board. The PAM_ODD_OUT* signal is the pattern clock divided by two, which turns on and off at the beginning and end of pattern play, respectively. It is sent to the PAM boards, where it is registered on the rising edge of the PCLK signal.

The VALID* signal is the Pattern Bus DATA_VALID* signal, before it is registered, level shifted, and placed on the Pattern Bus.

The BUF_EN* signal on line 780 is input to buffer control PAL 744. This signal, after further registering, is the signal that will be used for enabling the outputs.

Buffer control PAL 744 generates control signals which direct and gate pattern data during the presentation mode. The inputs to the PAL are the BUF_EN* signals on line 780, the RESET* signal on line 767, the PRESENT signal on line 524, and the WRITE* signal on line 759. The BUF_EN* signal is used to synchronously generate four state bits. These bits are routed internally to combinational logic, which then uses the remaining inputs to generate the four output signals.

PAL 744 outputs the BTL_DATA_OUT signal on line 781, the BTL_OUT signal on line 782, the PAT_ON* signal on line 783, and the CTRL_VALID signal on line 784. The BTL_DATA_OUT signal controls the direction of the four BTL transceivers used for the lower 32 pattern bits of backplane I/F 512. These bits differ from the remaining 48 pattern bits in that the lower bits are used as the shared address/data bus during access mode.

The BTL_OUT signal controls the direction of the bus transceiver of the backplane I/F. The transceiver receives the 5-bit PARITY signal and a 3-bit PEL_CTL signal. The BTL_OUT signal is asserted when the RESET* signal is asserted.

The PAT_ON* signal enables the registers of the backplane I/F used for the lower 32 pattern bits, the 5-bit PAM PARITY signal, and the 3-bit PAM PEL_CTL signal, and enables the BTL transceivers of the backplane I/F used for the upper 48 pattern bits. The PAT_ON* signal is asserted when the pattern-on state bit and the PRESENT signal are asserted and the RESET* signal is deasserted. As discussed, the CTRL_VALID signal is gated with the branch command bits and the stop bit PAM_STOP signal when pattern data is not valid. The PARITY_VALID signal is the registered CTRL_VALID signal. When the PARTY_VALID signal is asserted and the P_CHECK signal is asserted, the PRTY_ERROR signal is asserted. The error signal will disable the control word parity checker until the pattern data is valid. The CTRL_VALID signal is deasserted when the control-valid state bit is deasserted, the PRESENT signal is deasserted during the access mode, or the RESET signal line is asserted.

Request PAL 746 has asynchronous and synchronous sections. The asynchronous section combines CPU requests, synchronous transfer requests, and asynchronous transfer requests for generating the UNSYN_C_REQ* signal on line 613. This signal is sent to the request state machine. Request PAL 746 also has a synchronous section which is used only during pattern play. During this time, the request PAL generates six outputs which are the LOAD_CNTR* signal on line 786, NEXT_LINK* signal on line 932, and a 4-bit LOAD_VALUE signal on line 785.

Request PAL 746 has nine inputs; some of these signals have been previously described. The inputs to the request PAL were the RESET* signal on line 767, PRESENT signal on line 524, CPU_REQ* signal on line 552, BELOW_1 MHZ* signal on line 760, ASYN_C_XFER* signal on line 761, PAT_XFER* signal on line 756, STOP_REFRESH* signal on line 741, L_MUST_REFRESH* signal on line 743, and MUST_REFRESH signal on line 743. The RESET* signal

places the synchronous section of the PAL in a state in which the 4-bit **LOAD_VALUE** signal is asserted and each of the bits has a logic high value. This will allow the counter used by transfer PROM 748 to be loaded with a hexadecimal "F." The assertion of the **RESET*** signal also deasserts the **UNSYNC_REQ*** signal output from the PAL. Once **RESET*** is deasserted, the synchronous state machine goes to a wait state.

The **PRESENT** bit informs the asynchronous section of the request circuitry which mode the Pattern Bus is in.

The **CPU_REQ*** signal is asserted whenever the PAC/PAM is selected during access mode. When asserted, it will cause the **UNSYNC_REQ*** signal to be asserted.

During presentation mode, the **BELOW_1 MHZ*** signal determines whether the **ASYNX_XFER*** or the **PAT_XFER*** signal is used to generate the **UNSYNC_REQ*** signal. When the **BELOW_1 MHZ*** signal is asserted, the **ASYNX_XFER*** signal is used. Conversely, the **PAT_XFER*** signal generates the **UNSYNC_REQ*** signal when the **BELOW_1 MHZ*** signal is deasserted.

The synchronous section of the PAL always uses the **PAT_XFER*** signal. When the **PAT_XFER*** signal is first asserted, the synchronous section begins to count. In doing this, the **LOAD_VALUE** bits are used as a non-sequential counter. On the thirteenth count, the **LOAD_CNTR*** signal on line 786 is deasserted for one pattern clock cycle. This loads a seven into binary counter 750 that connects to transfer PROM 748. The request PAL then waits until the **PAT_XFER*** signal is deasserted before returning to a wait state which looks for the **PAT_XFER*** signal to be asserted. The remaining output of request PAL 746 is the **NEXT_LINK*** signal on line 932. This signal is the **PAT_XFER*** signal delayed one clock cycle.

The **STOP_REFRESH*** signal is the TTL version of the ECL **SAMPLE** signal. During HME sample periods, the **STOP_REFRESH*** signal is asserted and the **BELOW_1 MHZ*** signal is deasserted. The **UNSYNC_REQ*** signal output from the request PAL is asserted and clocked into the request circuitry synchronizer. The 2-bit **REQ_CODE** signal indicates a synchronous transfer request. This prevents the request circuitry from performing refresh cycles during the sample period.

The **MUST_REFRESH*** signal on line 743 is an output of refresh circuitry 510. If refresh cycles are being withheld, and more than 48 cycles are counted, the **MUST_REFRESH*** signal is asserted. This causes request PAL 746 to ignore the **STOP_REFRESH*** signal on line 741 and to deassert the **UNSYNC_REQ*** signal. The **L_MUST_REFRESH*** signal serves an identical function. It is necessary only when the **MUST_REFRESH*** signal is removed once refresh cycles begin to take place, even though refresh circuitry 510 has not processed all of the accumulated refresh clocks.

The transfer PROM 748 and its associated circuitry are used during presentation mode to provide various branch/transfer control signals. This circuitry includes PROM 748, 4-bit binary counter 750, and octal register 752. Binary counter 750 is normally disabled until it is loaded by request PAL 746 during a transfer operation. Once loaded with a seven, the binary counter is enabled and begins to count until it reaches a terminal count of hexadecimal "F." The output of the binary counter is the 4-bit **PROM_ADDR** signal that forms the lower

four address bits for PROM 748. The highest order address bit is the **BELOW_1 MHZ*** signal. This bit determines whether the PROM generates signals appropriate for synchronous transfer or asynchronous transfer. Waveforms are stored within PROM 748 and are output as the counter sequences through the PROM.

During synchronous transfers, when memory control signals are generated using the **P_CLK** signal, transfer PROM 748 outputs the **SYNC_RAS*** and **SYNC_CAS*** signals output for the register 752 on line 791 and 792, respectively, **SYNC_COL*** signal on line 795, and the **SYNC_EVEN_TR*** signal and **SYNC_ODD_TR*** signals on lines 793 and 794, respectively. During asynchronous transfers, the **ASYNX_XFER*** signal is asserted during the appropriate clock cycle. The **ASYNX_XFER*** signal is sent to request PAL 746.

During both types of transfer cycles, the transfer PROM circuitry generates the **XFER_FB*** signal on line 796 and the **NEXT_SAM*** signal on line 798. The **XFER_FB*** signal is a timing signal used by both transfer PAL 740 and pattern control PAL 742. The rising edge of the **NEXT_SAM*** signal registers the 2-bit **BD_ADDR** signal on line 749. This signal is from the link table circuitry. The **BD_ADDR** signal is registered and becomes the address for the next transfer operation. The **NEXT_SAM*** signal also loads the block offset counter with an FC, so that the count will be zero coincident with the first pattern resulting from the transfer.

FIG. 14 shows the elements of link table 506. The link table includes latches 902 and 904, RAMs 914 and 916, and bi-directional shift registers 926 and 928. The 16 bits from PAC address bus 906 are input to latches 902 and 908, with latch 902 receiving the upper order bits and latch 904 receiving the lower order bits. The **PASS_ADDRESS** signal on line 908 is input to the latch control inputs of latches 902 and 904. This active high signal indicates that no parity errors have been detected in the pattern control word. The **EN_LINK_ADDR*** signal on line 564 is input to the output enable inputs of the latches. This signal is one of the link table control signals output from decode logic 520 of the TMG/CPU I/F 502.

The output of the latches, except for the highest order bit, are placed on the link address bus 912. This consists of 15 of the 16 bits. The highest order bit on line 913 is input to bi-directional shift register 926 for placement on link data bus 924. The 15-bit address bus connects to the address inputs to RAMs 914 and 916. The signals input to the control inputs of the RAMs are the **WR_LINK*** signal on line 920, which is input to the write enable inputs, and the **EN_LINK_RAM*** signal on line 918.

The data inputs of RAMs 914 and 916 are connected to link data bus 924. The data bus also connects to bi-directional shift registers 926 and 928, such that the high order bits connect to register 926 and the low order bits connect to register 928. The inputs to the shift registers, in addition to the highest order address bit previously discussed, are the remaining 15 bits of link address bus 912.

The **CLK_BRANCH** signal on line 936 clocks the shift registers. The **CLK_BRANCH** signal is the output of AND gate 934. The signals input to AND gate 934 are the **WR_BRANCH*** signal on line 930 and the **NEXT_LINK*** signal on line 932. The **WR_BRANCH*** signal is one of the outputs of the PAC

write control decoder and the NEXT_LINK* signal is one of the outputs from the request PAL 746.

The PRESENT signal on line 937 and the RD_PRTY_ADDR* signal on line 938 are input to the control inputs of the shift registers. The PRESENT signal has been discussed. The RD_PRTY_ADDR* signal is one of the outputs from the PAC read control decoder.

The other control inputs to the shift registers are the READ_DATA* signal on line 546 and the EN_BRANCH* signal on line 940. The previously discussed READ_DATA* signal is input to the direction input. The EN_BRANCH* signal is one of the outputs from decoder logic 520 of TMG/CPU I/F 502, and is input to the output enable of the shift registers.

Now considering the circuit and its operation, isolated blocks of PAM memory are linked together by the link table. During the presentation mode, the 15-bit address bus that connects to shift registers 926 and 928, which are the branch registers, is routed to both the address generator which creates the PAM address for the next branch operation and to the 15-bit address of the link table memory.

Each time a branch occurs, the shift registers are clocked. In this manner, the link table data bits on line 924 become link table address bits on line 912, which look up the next branch address. This process is continued until pattern presentation is complete.

During the access mode, latches 902 and 904 allow the CPU to read from, and write to, RAMs 914 and 916, and branch shift registers 926 and 928.

Pattern Memory

FIGS. 15, 16, and 19 refer to the PAM circuitry and circuitry used for selecting specific PAM memory during the access and presentation modes.

Pattern memory 514, shown in FIG. 8, includes one or more individual PAM boards 516. Each PAM board is capable of storing 102-bit wide words, which are referred to as "patterns." Each of the individual PAM boards can store 128K, 512K, 2M, or 8M patterns (by 102 bits wide). The maximum memory depth of each PAM board depends upon the capacity of the VRAMs on it.

Referring to FIG. 15, analog comparator circuit 950 is shown. This circuit preferably is disposed on PAC 258. Its purpose is to determine the number of PAM boards that are connected to PAC 258. A resistor divider network 952 provides comparators 954, 956, 958, and 960 with a separate reference voltages. Nominally, these voltages are 2.50 v, 2.07 v, 1.48 v, and 0.55 v, respectively. The other input of each comparator is tied to a common signal, which in this case is the RESISTOR signal on line 962, and is a wire-tied composite of the RESISTOR signal from each PAM. Line 962 is connected to ground through 162 ohm resistor 964. Each PAM RESISTOR signal is based on PAM VCC after passing through a 562 Ohm resistor. As PAMs are added to the PAC, the 562 Ohm resistors are added in parallel and, therefore, the voltage seen at the input to each of the comparators increases. Each incremental change of voltage, caused by the addition of a PAM board, crosses one of the voltage thresholds. The nominal voltage of the comparator circuit input is 0 V for no PAMs, 1.12 V for one PAM, 1.83 V for two PAMs, 2.31 V for three PAMs, and 2.67 V for four PAMs. The comparator outputs are routed to a 4-bit buffer 966 which may be read during CPU access mode by asserting the RD_NUM_PAMS* signal on line 968. When

read, the output of buffer 966 is placed on the 4-bit PAM AD bus on line 970. The number of logic high values in buffer 966 indicates the number of PAMs on PAC 258.

FIG. 16 shows configuration PROM 1000 and its supporting circuitry. Configuration PROM 1000 serves as a look-up table. It has a 12-bit input and a 4-bit output. The input comprises two signals: the 6-bit CONFIG signal on line 1014, and the 6-bit LA (link address) signal on line 1012. The 6-bit LA signal is from the PAC row/column generation circuitry. The CONFIG signal is based on the number and size of the PAMs connected to PAC 258.

The output of PROM 1000 consists of a 2-bit BD_ADDR signal on line 1011 and a 2-bit MEM_SIZE signal on line 1018. BD_ADDR is input to two D flip-flops 1006 and 1008. These flip-flop are clocked by the NEXT_SAM* signal on line 1017. The output of the flip-flops is the 2-bit PAT_ADDR signal on line 1020.

BD_ADDR signal on line 1011 is multiplexed with the 2-bit REF_ADDR signal on line 1015 and 2 bits of the PAL address bus on line 1017. The output of multiplexer 1054 is the 2-bit PAM_ADDR signal.

The MEM_SIZE signal on line 1018 is input to the control input of shift register 1010. The signals determine the row/column selection for memory. An access to a 128K PAM requires a 2-bit shift. A 2M PAM would require a three bit shift.

On the basis of the 6-bit LA signal and 6-bit CONFIG signal, Configuration PROM 1000 determines which PAM board is being addressed and outputs the signal which corresponds to the appropriate board.

Each time pattern memory 514 is accessed, the 32-bit Pattern Bus address must be converted into a 2-bit board address and a multiplexed row/column address. Such a conversion must also be made to the link table address during branching operations. Both the link table address and 22 bits of the CPU address bus are sent to a multiplexer circuit. During CPU access mode, the CPU address bus is selected. During pattern presentation mode, the 15-bit link table address (appended with seven zeros as the least significant bits) is selected. The 6 most significant bits of this 22-bit multiplexed address bus are sent to configuration PROM 1000 (the LA signal). The 20 least significant bits are multiplexed onto a ten bit row/column address bus before being sent to the PAMs.

FIG. 17 shows an example of a pattern control word. Pattern words preferably are 102 bits wide including 6 parity bits, 80 data bits and 16 control bits which are shown at 1030 and 1032, respectively. There is one parity bit for every 16 data bits, and one parity bit for the 16 control bits. For HME devices wider than 80 pins, PACs in the multiple lanes are programmed with the same pattern control words and run synchronously.

Referring now to FIG. 18, control word 1032, which is part of each pattern in the sequence, includes 16 bits, illustratively labelled D0 through D15. Control word 1032 preferably includes the four PEL_ADDR bits at 1036 for addressing PEL 266 during pattern presentation. The three PEL_CONTROL bits 1038 are driven on the Pattern Bus during pattern presentation and are used by TG 248 and PELs 266. These bits control the function of the CICs on the PEL.

Control word 1032 preferably includes the USER bit at 1040. The bit may be driven to a test point on DAB 268. For this purpose, it is also driven onto the Pattern

Bus for receipt by the PEL during pattern presentation. This bit preferably is asserted throughout pattern presentation.

The STOP bit at 1042 preferably is provided to inform a PAC that a pattern sequence is complete. After the STOP bit is recognized, the PAC outputs three more patterns onto the Pattern Bus before halting. Therefore, the STOP bit must be stored three patterns ahead of the last pattern.

Control word 1032 preferably includes the 2 BRANCH bits at 1044. These bits are for controlling pattern-block branching during pattern presentation. The two bits may be set to one of four states, as shown in Table 4.

TABLE 4

D10	D9	BRANCH COMMAND
0	0	BRANCH NEVER
0	1	BRANCH ALWAYS
1	0	BRANCH ON FALLING FEEDBACK
1	1	BRANCH ON RISING FEEDBACK

"Branch never" causes the pattern sequence to continue sequentially without branching. "Branch always" causes unconditional branching to the link address. The remaining two commands cause conditional branching, conditioned by either the falling or rising edge of the FEEDBACK signals. Branching on the FEEDBACK signal is used to initialize non-resettable HMEs, as will be described.

The branch commands of "branch always" or "branch on feedback" must be specified 26 patterns before the last pattern to be presented in the current pattern block. Also, branch commands must be placed in "odd" numbered patterns in a pattern sequence because pattern memory is interleaved.

The remaining 5 bits of the control word at 1046 and 1048 preferably are spare bits, which are driven to the PAC or onto the Pattern Bus during pattern presentation.

PAMs are addressed by the CPU in three banks, each 32 bits wide. This memory organization requires that each pattern which is stored in a sequence must be stored across all three banks.

Division of the PAM is treated differently during access and presentation mode. During CPU access mode, pattern memory 514 is divided into the three banks, and each bank is further divided into a low data word and a high data word. During pattern presentation mode, however, the memory is divided into two banks, even and odd. Each "even" memory bank is paired with an "odd" memory bank. The serial output ports of each pair are connected to a multiplexer. The outputs of the various multiplexers are routed to data connectors. The connectors route the data to registers of PAC 258. This multiplexing allows the memories to be operated at one half the pattern presentation rate. The even and odd banks are clocked with complementary versions of the pattern clock.

FIG. 19 shows the PAM control circuitry located on the PAM. The PAM control circuitry includes a PAM decode PAL 1050, a PAM PAL 1060, two row address strobe ("RAS") PALs 1064 and 1066, column address strobe ("CAS") buffers 1068, a parity PAL 1080, and the data and parity memory banks 1070 and 1072.

The 2-bit PAM ADDR signal on line 1019 is input to PAM decode PAL 1050. The second input to PAL 1050 is the 2-bit STRAP signal from DIP switch 1052. The outputs of PAL 1050 include the PRTY_WRITE*

signal on line 1053, the PAM_SEC* signal on line 1055, PAM_ACCESS* signal on line 1051, and the PAM_READ* signal on line 1057. The PRTY_WRITE* signal is input to register 1074. The PAM_SEL* signal is input to RAS PALs 1064 and 1066 via lines 1065 and 1067, respectively. The PAM_ACCESS* signal enables registers that store data from banks 1070. The PAM_READ* signal is a control input to transceiver 1084. The READ_ID* signal is a control input to ID PROM 1056. Transceiver 1054 and registers 1074 control the transfer of the RC signal on line 1016 to the data and parity memory banks. The PRTY_WRITE* signal on line 1053 enables register 1074.

PAM PAL 1060 receives the STRAP signal from DIP switches 1052 and input signals from registers 1058 via line 1059. Line 1059 carries the registered PATT_ADDR and PAM_MUX_EN* signals, and the EVEN_XFER* signal.

The BD_SEL* output is looped back to registers 1058. The other outputs of PAM PAL 1060 are registered in register 1062 which is clocked by the BU-F_PCLK signal. The outputs of register 1062 are input to memory banks 1070 and parity memory 1072 on lines 1092 and 1094, and to the control inputs to multiplexers 1086 and 1088 on line 1090.

RAS PALs 1064 and 1066 control the high and low data words, respectively, for each of the three data memory banks 1070 and parity memory bank 1072. RAS PALs 1064 and 1066 receive the same input signals, except that PAL 1064 receives the H_WORD* signal, and PAL 1066 receives the L_WORD* signal. The other inputs to PALs 1064 and 1066 via lines 1065 and 1067 include: the EVEN_XFER* signal from register 1058, the PAM_SEL* signal from PAM decode PAL 1050, the RAS* signal, and various bank select signals. High and low CAS buffers 1068 receive buffered CAS* signals, and transmit them to the data and parity memory banks 1070 and 1072.

The inputs to parity PAL 1080 are the WS* signal on line 1083, the 2-bit SEL_BANK signal on line 1085, and the H_PARITY and L_PARITY signals on line 1081, which are outputs of H_PARITY multiplexer 1082. The outputs of parity PAL 1080 are the 3-bit H_PRTY signal and the 3-bit L_PRTY signal on line 1079. These outputs are input to parity memory 1072 and multiplexer 1082.

Multiplexer 1082 is controlled by the 2-bit SEL_BANK signal on line 1085. Multiplexer 1082 and transceiver 1084 are enabled by the PAM_READ* signal on line 1057 that is output from PAM decode PAL 1050.

Register 1076 and multiplexer 1078 transmit the buffered ODD_TR/OE* and EVEN_TR/OE* signals on line 1087. These signals enable the inputs of memory banks 1070 and 1072.

Referring to FIG. 20, as pattern sequences are created, pattern memory is allocated in 256-pattern blocks. The blocks need not be contiguous within the memory because link table 506 causes PAC 258 to branch from one block to the next, creating a continuous output. The first pattern in any sequence must be stored on a block boundary.

The following is a description of accessing pattern memory and its presentation. The patterns to be presented preferably are stored in VRAMs. Each of the VRAMs of the PAM contains a serial shift register. This register is as long as a row within the VRAM array

(256 bits for the VRAM used on the 128K PAM). The pattern data, when requested, is transferred to the VRAM shift register. Bits may then be shifted from this register at the preferred clock speed of 25 MHz. The 16-bit address specified in link table 506 defines the VRAM address of the next block to be transferred to the VRAM shift register. At the initiation of pattern presentation, a starting address in link table 506 must be specified for retrieving data from the VRAM.

Shift register transfer requests occur during pattern presentation. Any row within the VRAM may be transferred to the shift register by performing a shift register transfer cycle.

The row address selects the memory row that is transferred to the shift register. When the shift register transfer takes place, the column address selects one of the bits of the register as the first to be clocked out. If the shift register contents have all been clocked out and another shift register transfer cycle does not take place, the same data is output again. In other words, the shift register is cyclic.

PAC 258 performs a "feedback initialization" function in which non-resettable HME devices are placed in a known state. To do this, a stimulus pattern sequence is presented to the HME device until the desired state is attained. PAC 258 uses the shift registers inside the VRAM of the PAM to repeatedly present a given sequence of patterns to the device. A row of the VRAM is transferred into the shift register, where it is then continually clocked. Because the shift register is cyclic, the feedback pattern sequence is repeated indefinitely, as long as no branching takes place.

In the preferred embodiment of the invention, PAC 258 will stop presenting the feedback sequence once the device has achieved the desired state. To end feedback initialization and begin presenting normal patterns to the HME, a single-bit FEEDBACK* signal is sent from the HME through the TG to PAC 258. Either the rising or the falling edge of this FEEDBACK* signal is used to determine the state of the HME.

PAM And PAC Registers

As set forth previously, the PAC has been allocated 194 MBytes of CPU address space. The allocation is 192 MBytes to PAMs 514, 1 MByte for link table 506, and 1 MByte for the various control and status registers. The PAC has multiple control and status registers. One such status register is ID PROM 1056. Each PAM that is connected to the PAC also has an ID PROM, of which, ID PROM 1056 of FIG. 19 is representative. In addition, there are four PAM present status registers which are shared between the PAC and PAMs.

Various registers have been discussed in conjunction with the circuitry of which each is part. Here each will be discussed briefly to provide a reference section regarding these registers.

The PAC ID PROM 1056 (FIG. 19) is a 32-by-8 PROM which contains information pertaining to the PAC.

The number of PAMs register 966 is a 4-bit register which indicates the number of PAMs stacked onto the PAC. Table 5 shows the relationship between the number of PAMs connected to the PAC and the bit patterns found in the register. If the bit pattern is other than one of the five shown in Table 5, the PAC/PAM hardware is not functioning properly.

TABLE 5

D3	D2	D1	D0	No. of PAMs
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	1	1	1	3
1	1	1	1	4

Even though the all "0s" condition is legitimate, it is treated as an error condition since the PAC is not fully functional without at least one PAM.

The PAC configuration register 1102 (FIG. 16) is an 8-bit read/write register used to configure the PAC. Six of the bits are set according to the number and types of PAMs connected to the PAC. The 6-bit code is determined as shown in Table 6.

TABLE 6

A.	If only one PAM is connected to the PAC: CONFIG_CODE = 60 for a 128K PAM CONFIG_CODE = 61 for a 512K PAM CONFIG_CODE = 62 for a 2M PAM
B.	If more than one PAM is connected to the PAC: CONFIG_CODE = $[(\text{MEM_SIZE})/128] - 1$, where MEM_SIZE is the sum of the size of all PAMs

The two remaining bits configure the two parity generator/checker circuits. One bit configures the high word circuitry to compute either even or odd parity. The other bit configures the low word parity generator/checker in the same manner. Odd parity preferably is selected during normal operation.

The branch address registers 926 and 928 (FIG. 14) are 15-bit read/write registers used to specify the initial pattern address in block increments. Also, if a pattern sequence terminates due to an error condition, the CPU can examine the branch address register to determine which block was being played when the error occurred.

The error register is a 4-bit read/write error register which, together with a parity error address register indicates the source of an error condition. The CPU clears the error bit by performing a write operation. When the most significant bit is a logic high value, it indicates a refresh error has occurred. This is a fatal error. When the second most significant bit is a logic high value, it indicates a fatal hardware error. This error is generated by the state machine operating with the on-board 16 MHz clock. When the third most significant bit has a logic high value, it indicates that state machine operating the pattern clock has detected an error. This could be due to faulty hardware or an improperly loaded PAM. When the least significant bit has a logic high value, it indicates that a control word parity error occurred during a pattern presentation.

The block offset register is an 8-bit read only register which the CPU may read when an error occurs during presentation. It indicates the number of patterns offset from the block number residing in the branch register address.

The parity error address register is a 28-bit read only register containing the address of the memory location which generated a parity error. Two of the bits in the register indicate which word, low, high, or both, was in error. The CPU clears the two active-high error bits by performing a write operation on the error register. The two most significant bits are the high and low word parity error bits, respectively. The remainder of the bits

correspond to the address bits 27-2. These 26 bits uniquely specify the long word address in the PAM address space.

The PAM 0 present register is a single bit read only register which allows the CPU to determine whether any of the PAMs stacked on the PAC are associated with address 00. When the bit has a logic low value, it indicates that the PAM is present, and when it is a logic high value, it indicates that the PAM is not present.

The PAM 0 ID PROM is a 32-by-8 bit read only memory which contains information pertaining to the PAM associated with address 00, viz., memory size, revision number, etc.

Each of the other three PAMs has a similar PAM present register and PAM ID PROM.

Pin Electronics Circuitry

An overall description of the PEL and its operation will first be presented. PEL 266 connects to the Pattern Bus and provides the electrical interface to the HME. The HME connects to the PEL via the DAB. PEL 266 preferably supports 80 HME pins and includes five CICs.

During pattern-presentation mode, the PEL receives patterns from the Pattern Bus preferably at up to 25M patterns per second. These patterns are clocked into a register using the pattern clock from the TG. From there the patterns are clocked directly into the five CICs. Clocking into the CICs is preferably simultaneous. The HMS, for example, could alternatively clock patterns into the CICs sequentially to minimize Pattern Bus width.

When receiving patterns, the PEL compares a hard-wired 4-bit backplane slot address with the 4-bit PEL address on the Pattern Bus. The PEL accepts only those patterns addressed to its own slot. This allows patterns to be delivered sequentially to PELs in different backplane slots. Patterns can be data or control, based on the 3-bit PEL control signal on the Pattern Bus. Control patterns are used to load internal CIC control registers.

The PEL can be accessed by the CPU in access mode. The CPU uses the access mode to read from, and write to, certain PEL control registers and to read from, and write to, registers internal to the CICs on the PEL. All CPU reads and writes preferably are multiplexed on the 16-bit PEL data bus.

The PEL contains programmable reference-voltage generators which are initialized by the CPU in access mode. Four reference voltages are delivered to each of the five CICs to allow five-state sampling of the HME pins. One additional reference voltage is used as the target voltage ("VSL") for the low soft drivers of the five CICs on the PEL. One additional reference Voltage is used as the target voltage ("VSH") for the high soft drivers of the five CICs on the PEL. The PEL also contains a multi-channel analog-to-digital converter ("ADC") which can be read by the CPU in access mode to check the programmed reference voltages.

In addition to soft drivers, the five CICs on the PEL also contain hard and medium drivers which are used to drive HME pins. Most circuitry within the CIC is powered from the PEL +5 V power plane. However, the pins of the CICs which are connected within the CICs to VCC of the hard and medium drivers are not connected to the PEL +5 V power plane. Instead, these pins are connected to a special power plane of the PEL. This special power plane is separated from the +5 V power plane and is connected instead to the DAB

HMEVCC power plane. Thus, the hard and medium drivers of the CICs can drive between HMEVCC and signal ground, allowing HMEVCC on the DAB to be freely adjusted between +3 V and +5 V, independent of the +5 V PEL power plane.

Having presented an overall description of the PEL and its operation, the PEL will now be described in more detail.

PEL 266 in FIG. 1 is representative of all PELs of the system. Referring to FIGS. 21-24, the PELs of the system will be described. Generally, PEL 266 preferably includes one or more CICs which interface with the HME being modeled. PEL 266 preferably includes five such CICs, each of which provide 16 pins for interfacing with the HME. The five CICs are connected in parallel and, therefore, a total of 80 HME pins are supported by each PEL. Hence, HMEs having 80 pins or less may be connected to a single PEL using a DAB. In an alternative embodiment, the CICs are replaced with discrete circuits.

Referring now to FIG. 21, PEL 266 has circuitry for interfacing the CICs to the Pattern Bus. For descriptive purposes, the five CICs are shown at reference numbers 1100, 1102, 1104, 1106, and 1108. Each is mounted on PEL 266. Line 1144 from the Pattern Bus is input to BTL/TTL logic transceivers 1110, 1112, 1114, 1116, and 1118. A 16-bit signal is input to each of the five transceivers. An additional 16-bit control signal is transmitted from the Pattern Bus as part of the pattern data. Eight bits of the control signal are input to the PEL by the circuit shown at 1142.

In circuit 1142, the 8-bit signal is input to a BTL/TTL transceiver. The 8-bit output signal is input to an clocked octal register. When the register is clocked by the CONTROL* signal, the 8-bit signal is input to a series of PALs which will be described in detail in the discussion of FIG. 23. The outputs of each of the transceivers are input to clocked registers 1120, 1122, 1124, 1126, and 1128 and then to CICs 1100, 1102, 1104, 1106 and 1108, respectively.

During pattern presentation, data is sent from the PAM onto the Pattern Bus to the appropriate PEL. It is then processed by the BTL/TTL logic transceivers, and the clocked registers. The PCLK signal clocks registers 1120, 1122, 1124, 1126, and 1128, which output the data to the inputs of their respective CICs. The five CICs each output a 16-bit signal to the HME being modeled by five, 16-bit bi-directional lines which collectively are shown as the 80-bit bus line 1146.

The HMEs respond to stimulus signals sent over the 80-bit bus line, and the output is transmitted back to the CICs on bi-directional bus line 1146. When the CICs are read for determining the changes in the HME pin states as a result of the stimulus, the data is transmitted on the respective transceiver 1130, 1132, 1134, 1136, and 1138. The entire 80-bit signal is sent to the transceivers at the same time. The transceivers, however, are then sequentially enabled to multiplex the data onto a 16-bit address data bus and through transceiver 1140, as indicated by arrows 1148, onto line 1144 after processing by transceiver 1110.

Data is sent to the five CICs from PAC 258 and PAMs 514 on an 80-bit wide data bus. Data output from the five CICs is sent to CPU 240. It may then be sent to the host operating system to be used in simulations. The data path to the CPU is only 16 bits wide, so the data from the five CICs is multiplexed when transmitted from transceivers 1110, 1112, 1114, 1116, and 1118.

Transceiver 1140 changes operating states depending on whether the system is operating in pattern presentation mode or CPU access mode. The signal that controls this operation is the PLAY/CPU* signal on line 1154. During the CPU access mode, BTL/TTL transceivers 1112, 1114, 1116, and 1118 are not used. Instead, transceiver 1110 and transceiver 1140 are used. The transmission path between the Pattern Bus and various PEL control circuits in the access mode is via bus 1150 (shown in FIG. 22).

Referring now to FIG. 22, bus 1150 forms a 16-bit address/data bus between CPU 240 and various control and status registers on PEL 266. Control register 1174 of PEL 266 controls whether the HME is a public or private device, resetting error conditions, activating light-emitting diodes, and initializing DAB 268. The 8-bit signals output from control register 1174 are input to error PAL 1202 on line 1208, as described with respect to FIG. 25. The output of control register 1174 is also input to an electrically-erasable programmable memory chip (EEPROM) on the DAB which identifies the DAB that is attached to the PEL. The bits of control register 1174 are reset to a logic low value when the HMS is reset.

The PEL includes status registers 1176 and 1168. The inputs to register 1176 are the signals output from control register 1174. The signals input to register 1178 are signals from the CICs, the output of error PAL 1202, and the output of the EEPROM.

Digital-to-analog converters ("DACs") 1160, 1162, and 1164 are disposed on the PEL. The DACs generate the six voltages used by CICs 1100, 1102, 1104, 1106, and 1108. These voltages are the VLOGL signal on line 1173, the VLOGH signal on line 1175, the VLTH signal on line 1177, the VSL signal on line 1179, the VHTH signal on line 1181, and the VSH signal on line 1183. The CICs use these signals to generate reference voltages and currents for sampling and driving the HME pins.

DACs 1160, 1162, and 1164 are controlled by the CPU and are connected to address/data bus 1150. The reference voltages of the DACs are connected to HMEVCC. The HMEVCC signal is input to the DACs to minimize the danger of the DACs generating a voltage which exceeds the specifications of the HME.

The outputs from DAC 1162 and DAC 1164, associated with the VSL and VSH signals, respectively, preferably have an analog power buffer disposed in the line. The VSL signal on line 1179 is output from DAC 1162, and becomes the output signal of power buffer 1184. Similarly, the VSH signal on the 1183 is output from DAC 1164, then output from power buffer 1186. The signal shown at line 1204 represents the control signals for the DACs. The DAC output signals 1206 are transmitted to the CICs and to an ADC 1180. These ADC inputs serve the diagnostic function of verifying the outputs of DACs 1160, 1162, and 1164.

ADC 1180 is an octal converter. The inputs to the ADC include the outputs from DACs 1160, 1162, and 1164. The remaining inputs include the HMEVCC signal and the HME analog output. There is also a 5 V reference signal input to the ADC which is used for comparison to the HMEVCC signal. The ADC also connects to bus 1150.

Each PEL 266 has a distinct STRAP signal assigned to it. The STRAP signal must match the PEL address signal which is part of the 16-bit control word used during pattern presentation. If the PEL address signal

does not match the STRAP signal, the PEL is not selected for use. This will be explained in detail, referring to FIGS. 23 and 24.

In FIG. 23, comparator 1170 is connected to ADDR-/DATA bus 1150 via a clocked register 1168. This register is clocked by the AS* signal on line 1151. A 4-bit PEL address signal on bus 1150 is latched by register 1168. The other input to comparator 1170 is the 4-bit STRAP signal. The output of the comparator is the BOARDSEL signal on line 1218. This signal is asserted when the PEL address signal from register 1168 matches the STRAP signal.

The BOARDSEL signal is input to Auxiliary Decode PAL 1172, CIC Decode PAL 1171, and Miscellaneous PAL 1200. The 4-bit ADDR signal output from register 1168 on line 1220 and the PLAY/CPU* signal on line 1232 are input to PALs 1171 and 1172.

Auxiliary decode PAL 1172 outputs the IDPROM* signal on line 1211, CONTROL* signal on line 1210, and STATUS* signal on line 1213. The CONTROL* and STATUS* signals are input to status register 1178 of FIG. 22. The IDPROM* signal is input to chip select of the IDPROM. PAL 1172 also outputs the chip select signals for each of the DACs. These signals are shown on composite line 1204. Also output from PAL 1172 is the ADC* signal on line 1215, which is the chip select signal for ADC 1180.

The outputs of CIC decode PAL 1171 are the BUSEN* signal on line 1224, the CIC RD* signal on line 1226, the OUT/IN* signal on line 1228, and the 5-bit CIC DEN signal on line 1221. The BUSEN* signal enables each of transceivers 1130, 1132, 1134, 1136, and 1138. The OUT/IN* and CIC RD* signals are transmitted to each of the CICs of the PEL on lines 1228 and 1226, respectively.

In addition to the BOARDSEL signal, the inputs to Miscellaneous PAL 1200 include the CIC READ* signal, the ADCRDY signal on line 1223, the ADC signal on line 1215, the FEEDBACK signal on line 1225, and the BUSRESET* signal on line 1227. The CIC RD* and ADC signals were described previously. The ADCRDY signal is asserted once ADC 1180 has made a conversion and is ready to output the data. The FEEDBACK signal is input from the DAB connected to the PEL, and the BUSRESET* signal from the Pattern Bus.

The outputs of Miscellaneous PAL 1200 are the CIC AS* signal on line 1231, the DEVRESET* signal on line 1233, and the BTL_FBIN* signal on line 1229. The CIC AS* signal is a control input to each of the five CICs. The DEVRESET* signal controls a flip-flop, whose output is the previously discussed ACTIVE signal. The flip-flop is clocked by the previously described INITIALIZE signal. The BTL_FBIN* signal on line 1229 has been previously discussed regarding the PAC.

Error PAL 1202 receives as inputs an error signal from the CICs on line 1230 and the previously described signals output from control register 1174 on line 1208. The signals on line 1208 are the INITIALIZE signal, the ACTIVE signal, the LED signal, and the CTL RESET* signal. The remaining input signals are the DABPRESENT signal on line 1241, the PELEN* signal on line 1237, the AS signal on line 1153, and the PLAY/CPU* signal on line 1243. Error PAL 1202 is switched between pattern presentation mode and CPU access mode by the PLAY/CPU* signal. When an error condition is detected, error PAL 1202 will output

an ERROR* signal to output enable PAL 1232 on line 1212, and to status register 1178 on line 1210. Error PAL 1202 also transmits the BLT\$ERROR Pattern Bus error signal on line 1247 to the CPU. Another output of PAL 1202 is the INUSELED* signal on line 1235. This signal will cause a status LED to light when the PEL is being accessed or patterns are being played.

Referring now to FIG. 24, write control PAL 1194 and output enable PAL 1232 of control circuit 1142 are shown. The input signals to these PALs are registered signals from the Pattern Bus received through BTL/TTL transceiver 1190 and register 1192.

The inputs to transceivers 1190 are the 3-bit BTL\$PELCTL signal on line 1169, the BTL\$PELADDR signal on line 1170, and the BTL\$DATAVALID* signal on line 1171. The 8-bit output of the transceiver is input to register 1192. The output of the register are the 3-bit PELCTL signal on line 1173 and the 4-bit PELADDR signal on line 1175. These are input to write control PAL 1194. Write control PAL 1194 receives the 4-bit PELADDR signal during pattern presentation as part of the control word, compares it to the 4-bit STRAP signal 1216, and outputs a 2-bit SEL signal which is input to output enable PAL 1232. The PEL is "selected" if the PELADDR signal matches the STRAP signal. The 3-bit PELCTL signal is output from PAL 1194 as the WRCTL signal on line 1179. This signal is transmitted to the CICs and to output enable PAL 1232. If the PEL has been selected for operation, the signal will be asserted. The WRCTL signal is discussed in detail in connection with the integrated circuits.

Output enable PAL 1232 receives WRCTL and SEL signals from error PAL 1202, the ERROR* signal on line 1212, and the ACTIVE and PRIVATE/PUBLIC* signals from control register 1174 on line 1208. PAL 1194 also receives the DATAVALID signal on line 1177 from register 1192, and the PLAY/CPU* signal on line 1243. PAL 1232 outputs the SEQEND signal on line 1185 and the HMEOE* signal on line 1183 which are input to the CICs. The SEQEND signal is active high to initiate a measurement cycle at the end of a pattern sequence. The HMEOE* signal controls the HME pin drivers. The third output is the PELEN signal on line 1237.

PEL 266 generates an error signal when certain conditions occur. For example, PEL 266 generates an error signal if patterns are presented to the PEL before a DAB has been inserted, or when the PEL is not active. PEL 266 also generates an error signal when an error occurs on the CICs, such as shorting of the pins or a parity error. If the DAB is capable of live insertion, PEL 266 preferably generates an error signal when a DAB has been inserted, when the DAB is not active, and when a DAB has been removed after it has been initialized. Error signals relating to the DAB status inform the user that something has changed and that some action may be required.

Custom Integrated Circuits

An overall description of the CIC and its operation will first be presented. The CIC provides an electrical interface between the HMS and the HME pins. The CIC preferably is a mixed analog/digital CMOS standard-cell integrated circuit such as those manufactured by International Microelectronic Products. The CIC preferably supports 16 HME pins. The circuitry within the CIC for supporting a single HME pin is called a

"channel." All 16 channels of the CIC are identical in function.

Each CIC channel also has two internal single-bit pattern data registers. A "presentation" register holds the single-bit value that is currently being presented via the pin drivers of the channel. The other register is a "staging" register which is used in supporting HMEs with more pins than the width of the Pattern Bus. To support such HMEs, the staging registers of multiple CICs are loaded at different times, while all the CICs present constant pattern data from their pattern registers. After loading the staging registers of the final set of CICs, all the CICs are commanded to simultaneously transfer the contents of their staging registers to their presentation registers. Thus, the staging register allows patterns to be accumulated within multiple CICs in preparation for simultaneous presentation to a single HME.

The CIC channels have various programmable functions. They are software programmable on a per-pin basis via the Pattern Bus. The CPU programs the CIC channels by causing special control patterns to be presented to the PEL. Ordinarily, such control patterns are prefixed to the beginning of each pattern sequence stored in PAM. When the pattern sequence is presented, these prefix patterns initialize the CICs.

Each CIC channel includes an HME pin driver which connects to an HME pin via a series damping resistor and controlled-impedance traces on the DAB. The HME pin driver consists of a hard driver, a medium driver, and a soft driver. These three types of drivers have their outputs connected in parallel within the CIC. The hard driver is capable of driving very high current levels in either the high or low state. The medium driver preferably is current limited to less than 16 mA in the low state and less than 8 mA in the high state. The hard driver is used for HME pins which are strictly inputs (so that these HME pins transition quickly when driven by the CIC channel). The medium driver is used to drive HME I/O pins during the presentation of the history sequence. The reason for using the medium driver to present the history sequence is that the HME I/O pins may at times temporarily drive in the opposite direction than the CIC pin driver is driving, and the medium driver will limit the current in such a situation.

Because of the relatively low current limits of the medium driver, the HME pin will ordinarily dominate in a conflict with the CIC pin driver, but the resulting logic level is not always predictable. However, novel HMS software discussed elsewhere herein creates one-bit-per-pin patterns which cause the CIC channel to drive in the same direction as the HME pin. To do this, the HMS measures the state of each HME pin before each new pattern is added to the pattern sequence. If an HME pin is found to be driving during measurement, the new pattern is constructed so that the CIC will drive in the same direction as the HME pin. Therefore, the steady state for each pattern in the pattern sequence is that the HME and the CIC drive in the same directions, and any medium-driver conflicts are transient.

The soft driver is used during five-state measurement of the HME pin's state after presentation of a pattern sequence. All CIC's connected to the HME measure all HME pin states simultaneously after the HME outputs have settled. As previously discussed, the measurement process can distinguish driving low, driving high, non-driving low, non-driving high, and unknown ("H0,"

"H1," "Z0," "Z1," and "U") HME pin states. The result of each measurement is latched in a 3-bit register in the CIC channel.

The soft driver is implemented as two drivers connected in parallel inside the CIC: a low soft driver and a high soft driver. The low soft driver is enabled only if the corresponding pattern bit is a zero; it drives toward the soft-low target voltage ("VSL") programmed on the PEL. The high soft driver is enabled only if the corresponding pattern bit is a one; it drives toward the soft-high target voltage ("VSH") programmed on the PEL. The low soft driver and high soft driver have current limits which are separately programmable using control patterns presented to the CIC.

The low soft driver in each CIC channel is preferably programmed with a current limit such that if the connected HME pin is not driving (for example, if it is an I/O pin in the input state), then the low soft driver will succeed in driving it to a valid logic low state, but if the HME pin is driving, then the HME pin will dominate and the pin will attain the same logic level as if the CIC channel were not connected to it. In this case, we say that the CIC pin driver is driving "soft-low," or is driving with "soft-drive low I/V characteristic."

Similarly, the high soft driver in each CIC channel is preferably programmed with a current limit such that if the connected HME pin is not driving (for example, if it is an I/O pin in the input state), then the high soft driver will succeed in driving it to a valid logic high state, but if the HME pin is driving, then the HME pin will dominate and the pin will attain the same logic level as if the CIC channel were not connected to it. In this case, we say that the CIC pin driver is driving "soft-high," or is driving with "soft-drive high I/V characteristic."

Thus, a CIC channel driving "soft-high" or "soft-low" can be dominated by an HME pin which is driving, but will cause a non-driving HME pin to attain a valid logic high or low state, respectively.

During the measurement process, the CIC channel drives the connected HME pin using either a soft-drive low I/V characteristic or a soft-drive high I/V characteristic, depending upon the value of the appropriate pattern bit of the measurement pattern. The target voltage for the low soft driver is programmed to be slightly above the bottom of the logic low state voltage range for the HME pin. The target voltage for the high soft driver is programmed (on the PEL) to be slightly below the top of the logic high state voltage range for the HME pin.

While soft driving, the CIC channel compares the steady-state HME pin voltage simultaneously with four reference voltages generated on the PEL. The CIC channel may be soft-driving low or soft-driving high. If the CIC channel is soft-driving low, then an HME pin voltage lower than the target voltage of the low soft driver (VSL) indicates that the HME pin is in the H0 state. A voltage which is a valid logic low, but is above the target voltage, indicates that the HME pin is in the Z0 state. A voltage which is a valid logic high indicates that the HME pin is in the H1 state, and any other voltage indicates that the HME pin is in the U state. The Z1 state is impossible.

Similarly, if the CIC channel is soft-driving high, then an HME pin voltage higher than the target voltage of the high soft driver (VSH) indicates that the HME pin is in the H1 state; a voltage which is a valid logic high but is below the target voltage indicates that the HME pin is in the Z1 state; a voltage which is a valid

logic low indicates that the HME pin is in the H0 state; and any other voltage indicates that the HME pin is in the U state. The Z0 state is impossible.

Various alternative embodiments of this basic five-state sampling method would be obvious to one skilled in the art. For example, in an alternative embodiment, the current into, or out of, the HME pin could be compared with reference currents to distinguish the H0, H1, Z0, Z1, and U HME pin states.

In another alternative implementation, the CIC channel could be set to drive the HME pin toward a logic low level or a logic high level without regard to whether the CIC pin driver would dominate the HME pin. For example, the CIC channel could be set to hard drive the HME pin. As in the soft-drive method, voltage comparisons or current comparisons would be used to distinguish the H0, H1, Z0, Z1, and U pin states.

Now that its general operation has been presented, the CIC will be described in detail.

When simulator 164 of host computer 112 makes a request for a device evaluation, the HMS formats the request into signals to be presented to HME pins, applies the signals to the pins, measures the HME's behavior by sensing the pins after a predetermined time period, and returns the resulting outputs, plus timing information, to the simulator. An important consideration in this process is the method used to drive and sense the HME pins. In the present invention, CICs preferably are used to drive and sense the HME pins. In an alternative embodiment, some or all of these functions may be performed by discrete circuits. The CICs for the purpose defined herein may be packaged in any conventional packaging arrangement.

FIG. 25 at 1300, shows a chip-level block diagram of the CIC of the present invention. The major inputs to, and outputs from, CIC 1300 are shown. These are associated with system interface 1320, HME interface 1324, power supplies/references 1322, and local control 1326. Before disclosing the specifics of the components of CIC 1300 and their operation, the configuration of the CICs on the PEL will be disclosed.

Referring to FIG. 26 and 27, there are preferably five CICs connected to each PEL. Each one has sixteen HME driver/sensor pin channels. However, it is understood that there may be more or less than five CICs per PEL and each CIC may have a fewer or greater number of driver/sensor pin channels.

The five CICs may be configured in series or in parallel to increase the number of HME pin driver/sensors of the system. Preferably, the CICs are connected in parallel to allow pattern presentation at maximum rates, but the CICs could also be connected in series for lower cost. The design of an HMS based on serially-connected CICs is well within the capability of one skilled in the art. Any number of CICs may be connected together to accommodate an HME having more than 16 pins. FIG. 26 shows the parallel connection of CICs 1300, 1301 and 1303. CICs 1300, 1301, and 1303 are connected to their own 16-bit Pattern Bus, 1302, 1305, and 1307, respectively. They are also connected to their own 16-bit HME bus 1304, 1309, and 1311, respectively.

FIG. 27 shows CICs 1300, 1301, and 1303 connected in series. Each CIC is connected to its own 16-bit HME bus 1304, 1309 and 1311, respectively, but they all share the same 16-bit Pattern Bus 1302. When the CICs are connected in series, each CIC is sequentially enabled to access multiplexed pattern bus 1302.

When the system is connected in parallel, the speed of data presentation during pattern play is faster than when they are connected in series. This is because all the data bits for each CIC are sent from the PAYs at the same time. However, when they are connected in series, the multiplexed pattern bus serially loads all of the data bits from the PAMs. This method is slower.

When the system is series connected, the size of Pattern Bus 1302 is reduced. Further, when the CICs are serially connected, the system may include any desired number of CICs without the need to redesign the Pattern Bus and addressing system. Even further, to operate CICs 1300, 1301, and 1303 in parallel, each HME pin driver/sensor must be accompanied by its own pattern data line.

Again referring to FIGS. 25, CIC 1300 has a number of signals associated with it. Tables 7 and 8 provide an explanation of these signals which are referred to throughout the description of CIC 1300.

There are two modes that are used to perform the system interface function. The first is pattern presentation mode. In this mode, bit patterns containing control and data signals are clocked onto the Pattern Bus using the PCLK* signal. The patterns are presented to the pins of the HME. After the patterns have been presented, a measurement cycle is performed to evaluate the HME response to the pattern sequence. The second mode is CPU access mode. In CPU access mode, the Pattern Bus is used by the system to read data from, and write data to, the CICs, among other components. The following signals, listed in Tables 7 and 8, are used in both modes of operation as will be explained.

TABLE 7

Signal	Description
AS*	ADDRESS STROBE. This input is used during the CPU access mode. Asserting the AS* pulse causes PATTERN<5...2> to be latched into the internal address register of the CIC during the register read operation. AS* preferably is a TTL level input.
RD*	READ. This input is used during in CPU access mode. RD* enables reading of the CIC status registers onto the Pattern Bus. The internal address register of the CIC points to the status register being read. RD* preferably is a TTL level input.
WRCTL<2...0>	WRITE CONTROL. These inputs are used during pattern presentation mode to specify the operation the CIC will perform during the current pattern cycle. WRCTL<2...0> preferably are synchronous TTL level inputs.
PAT<15...0>	PATTERN I/O. During pattern presentation mode, the Pattern Bus is used to program the CIC control registers and to receive pattern data to be formatted onto the HME pins. During CPU access mode, the Pattern Bus is used to read the internal status registers of the CIC. PAT<15...0> preferably are synchronous TTL level I/O with weak pull-ups.
PARITY	PATTERN PARITY IN. PARITY is used during pattern presentation mode to check the parity on the Pattern Bus. If there is a parity error, the ERROR* signal is asserted and the data with the parity error is saved. The CIC parity is odd. PARITY preferably is a synchronous TTL level input.
EDGE<5...0>*	TIMING EDGES. The six timing edges are used by the data formatter as timing references during pattern pre-

TABLE 7-continued

Signal	Description
5	sensation mode to generate the data formats DNRZ, R1, RZ and RC for the HME outputs. These signals are preferably 8 ns high-going pulses. EDGE<5...0>* preferably are CMOS level inputs.
10 SAMPLE	SAMPLE. This signal is used during pattern presentation mode. The positive edge is used to take a 2-state sample of HME<15...0> for timing measurement, while the negative edge is used to take a 5-state sample during soft-drive sampling. SAMPLE preferably is a TTL level input.
15 PCLK*	PATTERN CLOCK. The falling edge of PCLK* is used in pattern presentation to clock data onto the synchronous input pins. PCLK* preferably is a CMOS level input.
20 IM	MEDIUM-DRIVER CURRENT SOURCE. This signal is used to generate a reference current for the HME medium-drivers. The medium-driver output currents at 2.5 V are $I_{\text{mol}} = 16 * V(\text{IM}) / R(\text{IM})$, and $I_{\text{moh}} = -8 * V(\text{IM}) / R(\text{IM})$. Preferably, the CIC will hold IM at VDD/2, and the R(IM) suggested value is 2.5K to VSS, which results in $I_{\text{mol}} = 16 \text{ mA}$ and $I_{\text{moh}} = -8 \text{ mA}$.
25 ISN	LOW SOFT-DRIVER CURRENT SOURCE. This signal is used to generate a reference current for the HME low soft-drivers. The formulas for the low soft-drivers are $I_{\text{sol}} < 3 > = 3 * V(\text{ISN}) / R(\text{ISN})$, $I_{\text{sol}} < 2 > = -1.2 * V(\text{ISN}) / R(\text{ISN})$, $I_{\text{sol}} < 1 > = 0.5 * V(\text{ISN}) / R(\text{ISN})$, $I_{\text{sol}} < 0 > = 0.2 * V(\text{ISN}) / R(\text{ISN})$. Preferably, the CIC will hold this pin at VDD/2, and suggested R(ISN) value is 2.5K to VSS, which results in $I_{\text{sol}} < 3 > = 3 \text{ mA}$, $I_{\text{sol}} < 2 > = 1.2 \text{ mA}$, $I_{\text{sol}} < 1 > = 0.5 \text{ mA}$ and $I_{\text{sol}} < 0 > = 0.2 \text{ mA}$.
30 ISP	HIGH SOFT-DRIVER CURRENT SOURCE. This pin is used to generate a reference current for the HME high soft-drivers. The formulas for the high soft-drivers are $I_{\text{soh}} < 3 > = -V(\text{ISP}) / R(\text{ISP})$, $I_{\text{soh}} < 2 > = -0.6 * V(\text{ISP}) / R(\text{ISP})$, $I_{\text{soh}} < 1 > = -0.25 * V(\text{ISP}) / R(\text{ISP})$, $I_{\text{soh}} < 0 > = -0.06 * V(\text{ISP}) / R(\text{ISP})$. The CIC will hold this pin at VDD/2, and the suggested R(ISP) value is 1.25K to VDD, which results in $I_{\text{soh}} < 3 > = -2 \text{ mA}$, $I_{\text{soh}} < 2 > = -1.2 \text{ mA}$, $I_{\text{soh}} < 1 > = -0.5 \text{ mA}$, and $I_{\text{soh}} < 0 > = -0.1 \text{ mA}$.
40 VHTH	Z1/H1 THRESHOLD. This voltage reference is used by the HME 5-state sampling circuitry. When the CIC is soft-driving an HME pin high, and $V(\text{LOGH}) < V(\text{HME}) < V(\text{HTH})$, then the Z status bit for the HME pin will be sampled TRUE. If $V(\text{HME}) > V(\text{HTH})$, then the Z status bit would be FALSE.
45 VLOGH	U/Z1 THRESHOLD. This voltage reference is used by the HME 5-state sampling circuitry. If $V(\text{HME}) > V(\text{LOGH})$ then the HME pin will be sampled as a H1 or Z1. If $V(\text{LOGH}) < V(\text{HME}) < V(\text{LOGL})$, then the HME pin will be sampled as a U.
50 VLOGL	ZO/U THRESHOLD. This voltage reference is used by the HME 5-state sampling circuitry. If $V(\text{HME}) < V(\text{LOGL})$, then the HME pin will be sampled as a logic low. If $V(\text{LOGL}) < V(\text{HME}) < V(\text{LOGH})$, then HME pin will be sampled as a logic "U."
55 VLOGH	
60 VLOGL	
65 VLTH	H0/ZO THRESHOLD. This voltage reference is used by the HME 5-state sampling circuitry. When the CIC is

TABLE 7-continued

Signal	Description
VSH	soft-driving an HME pin low, and $VLTH < V(HME) < VLOGL$, then the Z status bit for the HME pin will be sampled TRUE. If $V(HME) < VLTH$, then the Z status bit would be FALSE. HIGH SOFT-DRIVE POWER. This is the high soft-driver power supply. Its input range is from 2.5 volts to HMEVCC.
VSL	LOW SOFT-DRIVE POWER. This is the low soft-driver power supply. Its input range is from 0 V to 1.0 V.
AVDD	ANALOG POWER. This is the analog +5 V supply.
VDD	POWER. The VDD pins are for a +5 V power supply. There are two pins for pattern I/O, and one pin for internal logic.
VSS	GROUND. The VSS pins are for the ground reference. There are four pins for pattern I/O, two pins for internal logic, and two pins for analog cells.

Local control deals primarily with error handling. A preferred method of handling errors is to use the ERROR* signal to deassert the HMEOE* signal, and send an interrupt back to the HMS. The ERROR* signal preferably is also sent to a local status register on PEL 268. HMEOE* should only be asserted while patterns are being presented to an HME. For private devices, this occurs during the entire time the device is being used; for public devices it is only during individual pattern sequences. These signals are indicated generally at 1326 in FIG. 25. The last two groups of I/O signals of CIC 1300 interface the CIC with the HME and power the CIC. These signals are designated generally by reference numerals 1322 and 1324. These signals and their descriptions are set forth in Table 8.

TABLE 8

Signal	Description
ERROR*	ERROR. ERROR* is an open drain output that indicates there was either a parity error on the Pattern Bus during pattern presentation, or a HME pin was detected to be shorted. The current input pattern is saved when this pin goes low. To reset this signal, a read of the reset register is required. ERROR* is a CMOS level I/O.
HMEOE*	HME OUTPUT ENABLE. Deasserting HMEOE* disables the HME drivers, and enables a weak pull-down. HMEOE* preferably is a TTL level input.
HME<15..0>	HME I/O. The HME bus is used to drive and sense HME pins. HME<15..0> preferably are programmable level I/O pins.
HMEVCC	HME POWER. This signal is the HME I/O pin-driver power supply. The advantage to having a separate supply from the VDD pins is that HMEVCC can be from 3 V to 5 V, allowing for the direct driving of low-voltage CMOS parts.
VSS	GROUND. There are four ground pins allocated for HME pin drivers. These ground pins should be common to both the CIC and the HME.

CIC 1300 includes both digital and analog electronic circuitry. The digital circuitry includes data channels, control and data registers, and formatting circuitry. The analog circuitry includes the HME driver circuitry and 5-state sampler. The following description relates to the

preferred embodiment of the present invention which includes CICs that have 16 HME pin driver/sensor channels.

CIC 1300 has the capability of per-pin programmability for driving and sensing the HME pins. Hence, each of the CIC's HME pins can be programmed as an input, an output, or an I/O. In addition, each pin may be programmed as a power or ground pin so that the user can jumper power or ground to the HME without having to disconnect the HME pin. The CIC is protected against damage when HME pins are connected to power or ground.

Referring to FIG. 28, a block diagram of CIC 1300 shows the major components of the CIC. CIC 1300 preferably includes 16-bit Pattern Bus 1302 and 16-bit HME bus 1304. Data channels 1350 process the Pattern Bus signals for output therefrom to the HME bus. In processing data, data channels 1350 format data from the Pattern Bus and drive the formatted data on to the HME bus for input to the HME pins. The data channels sample the HME pins.

Circuit 1300 includes address latch 1352 which latches four pattern bits $PAT<5.2>$ in a 4-bit register. The latched address is transmitted to each of the data channels 1350, to be used as a status/read select. When RD* on line 1317 is asserted, the internal RESET signal on line 1319 of circuit 1300 is asserted. The RESET signal is input to parity checker 1354 and data channels 1350. The latched address values are input to data channels 1350, as the 4-bit ADDR signal on line 1321 when the AS* signal on line 1323 is asserted.

CIC 1300 also includes a parity checker 1354, an error handler 1358, and control circuitry 1356. The parity checker will now be described.

During pattern presentation, parity checker 1354 (FIG. 29) checks Pattern Bus 1302 for parity against the PARITY signal on line 1329. The 16-bit input connects to eight exclusive-OR gates, two adjacent bit lines per gate. The output of the gates are input to four more exclusive-OR gates, two adjacent bit lines per gate. This process is continued until there is a single output signal. The four levels of exclusive-OR gates are represented by the exclusive-OR gate 1309. The signal output of gate 1309 is output to exclusive-OR gate 1310. The second input signal is the registered PARITY signal output from register 1332.

The PARITY signal on line 1329 is from the Pattern Bus and latched in register 1322. The PARITY signal is from the PAC. The register is clocked by PCLK signal on line 1361. The control inputs to the register are the WRITE* signal on line 1331 and its complement.

The output of exclusive-OR gate 1310 is input to OR gate 1314. The second input to OR gate 1314 is the signal on feedback line 1333. The output of OR gate 1314 is input to the data input of register 1312. The output of register 1312 is the PAROUT signal on line 1333 which is also fed back to the input of OR gate 1314. The control inputs to register 1312 are the output of AND gate 1318 and its complement.

The inputs to AND gate 1318 are the ERROR* signal on lines 1328 and the output of register 1316. The WRITE signal on line 1331 is in input to the data input to register 1316. PCLK signal on line 1361 in input to its clock input.

The PARITY signal output from register 1322 is input to the data input of register 1320. This register is clocked by the PCLK signal and its control inputs are

the ERROR* signal on line 1326 and its complement. The output of the register is the PARIN signal on line 1335.

The PARIN signal line 1335 is the twice registered PARITY signal. This signal is clocked with the PAROUT signal so that if there is a parity error, the PARITY signal which generated the error may be retrieved. The PAROUT signal on line 1333 is a comparison of the PARITY signal on line 1329 and the PARDATA on line 1327.

If an error is detected, register 1312 and OR gate 1314 form a "hold" loop. This loop is controlled by the status ERROR* and WRITE signals, as gated by AND gate 1318. This loop will remain set by the RST* signal on line 1319.

Referring to FIG. 30, error handler 1358 will be described. If an error exists on the CIC, viz., parity or short error, it is preferably sent to all data channels 1350 by error handler 1358. The PAROUT signal on line 1333 and the SHORT signal on line 1337 are input to OR gate 1330. The SHORT signal when asserted indicates that there is a short in the data channel circuitry. The output of OR gate 1330 enables a tri-state buffer 1332. This will assert the PERROR* signal on line 1334 which is input to the PEL, and assert the ERROR* signal on line 1328 which is input to parity checker 1354 and data channels 1350.

The ERROR* signal on line 1328 is sent to the data channels so that the error data can be held for future evaluation. The PARIN signal on line 1335 is saved in status register 1320. Error handler 1358 forces the data channels and parity checker 1354 into the "hold" state whenever the ERROR* signal is asserted (either internally or externally).

FIG. 31 shows a block diagram of control circuitry 1356. Control circuitry 1356 generates certain control signals during pattern presentation mode. Certain of these signals control the input to the data channels. The single bit WRITE signal on line 1331 is sent to parity checker 1354 indicating when the parity checker has valid data. The circuit of FIG. 31 will now be discussed in detail.

The PSEQEND* signal on line 1341 is input to the data input of flip-flop 1347. The flip-flop is clocked by the PCLK signal. The SEQEND signal output from flip-flop 1347 on line 1349 is input to data channel 1350.

The 3-bit WRTCL signal on line 1179 is from PEL 266. The 3 bits are input to NOR gate 1343. The output of NOR gate 1343 is input to the data input of flip-flop 1345. This flip-flop is also clocked by the PCLK signal on line 1361. The output of the flip-flop 1345 is the PELEN signal on line 1351 which is input to the data channels, 1350.

The WRITE signal on line 1331 is the WRCTL<2> bit of the 3 bit WRCTL signal. This signal is input to parity checker 1354.

A 2-bit WRCTL<1..0> signal that is part of the 3-bit WRTCL signal on line 1179 is input to OR gate 1355. The output of OR gate 1355 is input to the data input of flip-flop 1355 whose output is clocked by the PCLK signal. The output of flip-flop 1355 is the load formatter ("LDFMT") signal on line 1357. The LDFMT signal is input to data channels 1350.

FIG. 32 is a block diagram of data channel 1350. It includes data and control registers 1370, status/read control circuitry 1372, edge selector 1374, formatter/driver circuitry 1376, and sampler 1378. Registers 1370 receive data from the Pattern Bus. This data is stored in

either a configuration register (FIG. 16), a data register, or the hard/medium drive enable register.

Status/read control 1372 is a 16:1 multiplexer that receives inputs from various internal nodes, including registers 1370, formatter/driver 1376, and sampler 1378.

Edge selector 1374 selects which of six timing edges will be used as the "set" edge, the "reset" edge, and the "driver off" edge for a particular bit of pattern data.

The inputs to formatter/driver circuitry 1376 are from data and control register 1370 and edge selector 1374. These inputs cause the formatter/driver circuitry to format the pattern data and drive it to the HME pins.

Sampler 1378 preferably performs two-state sampling on the rising edge of the SAMPLE signal, and five-state sampling on the falling edge of the signal. Sampler 1378 also includes circuitry for detecting short circuits. Each of these components is described subsequently.

Data and Control registers 1370 include configuration register 1390, shown in FIG. 33, and data and HMDEN* (hard/medium drive enable) registers 1410 and 1412, shown in FIG. 34. Configuration register 1390 is an 11-bit shift register with a parallel-load shadow register. The configuration register must be fully updated when it is written because of the shifting operation. The data and HMDEN* registers, however, may be individually updated in each pattern.

Referring to FIG. 33, configuration register 1390 includes 11-bit shift register 1392 and 1-bit shadow register 1394. The combination of these registers provide control signals for the data channels. The signals output from the configuration register 1390 are set forth in Table 9 and Table 10. The signals in Table 9 are output from shift register 1392 and the signals in Table 10 are output from shadow register 1394.

TABLE 9

Signal	Description
SEQHD*	IN SEQUENCE HARD-DRIVE ENABLE. Enables hard-driving during the interior of a sequence (SEQEND = FALSE) when HMDEN* is asserted low. Occurrence of a SET or RESET edge turns on the hard-drive output enable.
SEQMD*	IN SEQUENCE MEDIUM-DRIVE ENABLE. Enables medium-driving during the interior of a sequence (SEQEND = FALSE) when HMDEN* is asserted low. Occurrence of a SET or RESET edge turns on the medium-drive output enable.
LCYCHD*	LAST CYCLE HARD-DRIVE ENABLE. Enables hard-driving during the last pattern cycle (SEQEND = TRUE) when HMDEN* is asserted low. Occurrence of a SET or RESET edge turns on the hard-drive output enable.
LCYCMD*	LAST CYCLE MEDIUM-DRIVE ENABLE. Enables medium-driving during the last pattern cycle (SEQEND = TRUE) when HMDEN* is asserted low. Occurrence of a SET or RESET edge turns on the medium-drive output enable.
HDDIS*	HARD-DRIVER DISABLE. When HDDIS* is asserted low, then the DOFF edge turns off the hard-driver during both the interior of a sequence and the last cycle of a sequence.
DOFF	DRIVER OFF EDGE SELECT. The DOFF edge turns off selected hard-drivers (using HDDIS*), and it turns off selected medium-drivers during the last cycle when LCYCMD* is asserted low.
SET<1..0>	SET EDGE SELECT. Selects the edge for setting HME formatted data to a logic high value.

TABLE 9-continued

Signal	Description
RESET<1..0>	RESET EDGE SELECT. Selects the edge for setting HME formatted data to a logic low value.
CTL_OUT	CONTROL REGISTER OUTPUT. Performs no internal CIC control. Its purpose is to match the length of the shift register with the parallel load shadow register. It is also the bit of the configuration register that can be read (address 3).

TABLE 10

Signal	Description
SDHEN<3..0>	SOFT-DRIVE HIGH ENABLE. Selects the current for soft-driving a logic high value during the last cycle (SEQEND = TRUE).
SDLEN<3..0>	SOFT-DRIVE LOW ENABLE. Selects the current for soft-driving a logic low value during the last cycle (SEQEND = TRUE).
TOGGLE	TOGGLE FORMAT. When TRUE, TOGGLE selects the Input Pattern Data to be toggled when PAT<X> = 1, and unaffected when PAT<X> = 0. When FALSE, PAT<X> Input Pattern Data is unaffected.
FORMAT<1..0>	FORMAT: selects the data format used for a channel (RC = 00, RZ = 01, R1 = 10, DNRZ = 11).

The method by which these signals are output from configuration register 1390 will now be described.

11-bit shift register 1392 includes eleven flip-flops, 1391A–1391K. Each flip-flop has two data inputs and a single output. For example, the first data input of flip-flop 1391A is connected to patten bus 1302. The output of flip-flop 1391A is connected to the first data input of the flip-flop 1391B. This output-to-input connection method between successive flip-flops is repeated for the remainder of the flip-flops, 1391C–1391K. The first input is provided at the output when the SHIFT signal on line 1393 is asserted and the second is provided at the output when the SWAP signal on line 1395 is disasserted. Flip-flops 1391A–1391K are clocked by the PCLK* signal on line 1399.

Parallel load shadow register 1394, like shift register 1392, has eleven flip-flops, 1397A–1397K. These flip-flops, however, have one input and one output and are enabled when the SWAP signal on line 1395 is asserted. Flip-flops, 1397A–1397K are also clocked by the PCLK* signal on lines 1399.

The output of each flip-flop of shift register 1392 is connected to the data input of one of the flip-flops of shadow register 1394. The output of each shadow register flip-flop is connected to the second data input of the flip-flop of shift register 1392 to which its data input is connected to the output of that same shift register flip-flop.

In operation, data from Pattern Bus line 1302 that is properly addressed is serially input to shift register 1392. This is accomplished by asserting the SHIFT signal which enables the first data input of each of the eleven flip-flops, 1391A–1391K. Therefore, on the next eleven PCLK* clock pulses, the eleven bits on the pattern data are serially loaded into shift register 1392.

The data that is loaded is control data that is organized such that the SEQHD* signal is at first flip-flop 1391A (or first bit) and the CTL_OUT signal is at

eleventh flip-flop 1391K (or eleventh bit). The remainder of the flip-flops are loaded with a control data in the order presented in Table 9.

After shift register 1392 is loaded, the SHIFT signal is deasserted and the SWAP signal is asserted. This enables flip-flops 1397A–1397K of the shadow register and enables the second data input of the flip-flops 1391A–1391K. On the next clock, the data at the output of the flip-flops 1391A–1391K is clocked through shadow register flip-flops 1397A–1397K.

During this same clocking operation the data at the output of the flip-flops 1397A–1397K, which is presented at the second data input of shift register flip-flops 1397A–1397K, is clocked through these flip-flops. This data will be shifted out as the next 11 bits are shifted into shift register 1392.

FIG. 34 shows the data and HMDEN* registers of data channels 1350. The data register is shown at 1410 and the HMDEN* register is shown at 1412. The Pattern Bus signal on line 1302 is input to the data input of each register and both registers are clocked by the PCLK signal on line 1361. However, each register is enabled by a different signal. Data register 1410 is enabled by the WRITE signal on line 1331 and the HMDEN* register is enabled by the LDHMDEN* (Load hard/medium drive enable) signal on line 1411.

The output of data register 1410 is input to toggle logic 1417. The other inputs to toggle logic 1417 are the TOGGLE signal on line 1419 and the feedback signal on line 1421 from the output of formatter register 1414. Toggle logic 1417, based on the state of the TOGGLE signal 1419, will provide at the output, the fed-back FOMDATA (formatted data) signal on line 1415, or the output of the data register 1410.

The output of the toggle logic is input to the data input of formatter register 1414. This register is clocked by the the PCLK signal on line 1361 and is enabled by the LDFMT signal on line 1357. The output of formatter register 1414 is the FOMDATA signal on line 1415 which is used at various places in the circuitry as will be disclosed.

The output of HMDEN* register 1412 is input to the data input of formatter register 1416. This formatter register is clocked by the the PCLK signal on line 1361 and is enabled by the LDFMT signal on line 1357. The output of formatter register 1416 is the HMDEN* signal on line 1413 which is also a signal used in various places in the circuitry.

The WRITE signal and the LDHMDEN* signal are asserted at different times so that the data from each register can be loaded into its associated formatter register on different patterns and output from the formatter registers at the same time when the LDFMT signal is asserted.

With regard to serially configuring the CICs, as shown in FIG. 27, the data and HMDEN* registers of all of the CICs are serially loaded with pattern data. Once each has its data clocked to its respective formatter registers, all of the formatter registers are enabled at the same time with the LDFMT signal.

FIG. 35 shows edge selector circuitry 1374 of data channels 1350. Each data channel 1350 includes edge selector 1374. Generally, edge selector circuitry 1374 allows the user to program which of the six timing edges will function as the set, reset, and driver off edges. The six timing edges are generated by timing generator 248 and input to edge selector 1374 via the Pattern Bus.

The four inputs to the edge selector circuitry are the 2-bit SETSEL signal on line 1431, the DOFFSEL signal on line 1451, the 6-bit EDGE signal on line 1325, and the 2-bit RESETSEL signal on line 1485. All of these signals except the 6-bit EDGE signal are output from shift register 1392 (Table 9). A control input to the circuitry is the PELEN* signal on line 1351.

The 6-bit EDGE signal on line 1325 is a 6-bit bus that connects to data selectors 1439, 1461, and 1471. From this bus, the EDGE 2, 3, 4, and 5 bits are input to data selector 1439, the EDGE 0, 1, 4, and 5 bits are input to data selector 1461, and the EDGE 0, 1, 2, and 3 signals are input to data selector 1471.

The 2-bit SETSEL signal is input to decoder 1433. The decoder provides a parallel, 4-bit output on bus 1435. Bus 1435 is connected to the four enable inputs of data selector 1439. Based on the logic values of the 4-bit signal from bus 1435, one of the EDGE signals is selected as the ESET signal on line 1443 after being processed by inverter 1441.

The DOFFSEL signal on line 1451 is input to NAND gate 1455 after being processed by inverter 1453, and is input to NAND gate 1459. The PELEN signal on line 1351 is the second input to NAND gates 1455 and 1459 after being processed by inverter 1457. The outputs of the NAND gates are tied and input to the enable inputs of data selector 1461. The selected EDGE signal is the output from the data selector and is the DOFF signal on line 1465 after it has been processed by inverter 1463.

The 2-bit RESETSEL signal is input to decoder 1467. Decoder 1467 has a parallel, 4-bit output that is put on 4-bit bus 1469. Bus 1469 is connected to the four enable inputs to data selector 1471. The logic values of the 4-bit signal input to the enable inputs cause selection of one of the EDGE signals as the ERES signal on line 1477 after being processed by inverter 1475.

The PELEN* signal is also the output enable signal for decoders 1433 and 1467. It is also the gate signal for the p-channel devices used as pull-ups for each of the output lines of the edge selector.

Table 11 shows the preferred edge select options and the corresponding edge assignments. However, it is understood that other selection options and edge assignments are possible within the scope of the present invention.

TABLE 11

Edge Type	Select	Output
Set EDGE	00	EDGE<2>
Set EDGE	01	EDGE<3>
Set EDGE	10	EDGE<4>
Set EDGE	11	EDGE<5>
Reset EDGE	00	EDGE<0>
Reset EDGE	01	EDGE<1>
Reset EDGE	10	EDGE<2>
Reset EDGE	11	EDGE<3>
DOFF EDGE	00	EDGE<0>
DOFF EDGE	01	EDGE<5>

An important feature of edge selector 1374 is that it can disable any timing edge from passing through the edge selector output, thereby permitting an entire PEL 266 to be disabled.

The data selectors of edge selector circuitry 1374 may be designed to suit the needs of particular applications. In applications in which it is desirable to save die area on the CIC, two 4:1 data selectors (with a 2-bit

select) and one 2:1 data selector (with a 1-bit select) may be used. Where die area is not a limiting factor, it may be preferable to use three 6:1 data selectors or even three 8:1 data selectors. However, additional pins may be required to address the larger data selectors.

FIG. 36 shows data formatter 1376. Formatter 1376 includes hard-drive output enable formatter 1440, medium-drive output enable formatter 1442, data formatter 1444, and soft-drive output enable formatter 1446. Formatter 1376 also includes hard driver 1448, medium driver 1450, and soft driver 1452.

The common inputs to hard-drive output enable formatter 1440, medium-drive output enable formatter 1442, and data formatter 1444 are the ERES signal on line 1477 and the ESET signal on line 1443. A common input to just the hard-drive output enable formatter and the medium-drive output enable formatter is the DOFF signal on line 1465. The three signals, ERES, ESET, and DOFF, relate to the timing edges that were selected. The other inputs to the hard-drive output enable formatter are control signals that are representatively shown as the single input signal CONTROL. The signals on this line are the LCYC* signal on line 1495, HMDEN* signal on line 1413, HDDIS* signal on line 1515, SEQHD* signal on line 1513, LCYCHD* signal on line 1511; and the HMEOE* signal on line 1517 (FIG. 37). The output of the hard-drive output enable formatter is the HDOE signal on line 1501 which is the output enable signal for hard driver 1448.

Similarly, the control inputs to medium-drive output enable formatter are the LCYC* signal on line 1495, SEQMD* signal on line 1521, HMDEN* signal on line 1413, LCYCMD* signal on line 1519; the HMEOE* signal on line 1517 (FIG. 38). The output of the medium-drive output enable formatter is the MDOE signal on line 1503 which is the enable signal for medium driver 1450.

The other inputs to data formatter 1444 are the 2-bit FORMAT signal on line 1523 and FOMDATA signal on line 1415. The output of data formatter 1444 is the FMTOUT (formatted data) signal on line 1507 which is input to the hard and medium drivers, and the soft-drive output enable formatter.

The inputs to soft-drive output enable formatter 1446 are the 4-bit SDHEN signal on line 1491, the 4-bit SDLEN* signal on line 1493, the LCYC signal on line 1489, which is the inverted LCYC* signal, and, as stated, the FMTOUT signal on line 1507.

Data formatter 1376 uses the ESET signal to set HME pins to logic high value, the ERES signal edge to reset HME pins to logic low value, and the DOFF signal to turn off the hard and medium drivers. The set and reset edges are also used to turn on the hard and medium driver. However, even though the set edge changes data from a logic low to a logic high value, it does not affect data which is already a logic high value. Conversely, even though the reset edge changes data from a logic high to a logic low value, it does not affect data which is already logic low value.

FIG. 37 is a block diagram of hard-drive output enable formatter 1440 in greater detail. The signals input to hard-driver output enable formatter were described in discussing FIG. 36. Formatter 1440 includes hard-drive control circuitry 1460, SR latch 1462, and AND gate 1464. The control circuitry operates according to the true table set forth in Table 12.

TABLE 12

HMDEN*	LCYCHD*	SEQHD*	HDDIS*	LCYC*	EDGE(s) needed	LATCH
1	X	X	X	X	X	RESET
X	X	X	0	X	DOFF	RESET
0	X	0	X	0	Set or Reset	SET
0	0	X	X	1	Set or Reset	SET

The input signals to HDOE formatter control 1460 will cause it to generate output signals at the set and reset outputs which are input to SR latch 1462. The output of the SR latch is input to AND gate 1464. The second input to AND gate 1464 is the HMEOE* signal on line 1464. This signal is input to the negative-true input of the AND gate. The output of AND gate 1464 is the HDOE signal on line 1501.

According to the logic in Table 12, several specialized operations are possible. Assertion of the LCYC* signal permits the HMS to hard drive an HME pin at one stage of the pattern presentation (e.g., during the "restore" operation of pattern presentation) and to soft drive the same HME pin at a second stage of the simulation (e.g., during the "measurement" cycle of pattern presentation).

HME pins known to be dedicated input pins preferably are always hard-driven to establish a fast, clean transition from a logic high to logic low value and from a logic low to logic high value. There is no need to soft-drive such pins since these pins can never become output pins.

HME I/O pins can be driven by pulsing the hard driver. Circuit 1460 also enables operating with two bits per pin (normal operation is one bit per pin). When operating in the two bits per pin mode, the first bit is control bit, and the second bit is the data bit.

FIG. 38 shows medium-drive output enable formatter 1442 in greater detail. The signals input to the medium-driver output enable formatter were described in discussing FIG. 38. The components of formatter 1442 are similar to the components of hard-drive output enable formatter 1440. The medium-driver output enable formatter includes medium-drive control circuitry 1470, SR latch 1472, and AND gate 1474. This circuitry operates according to the truth table set forth in Table 13.

TABLE 13

HMDEN*	LCYCMD*	SEQMD*	LCYC*	EDGE(s) needed	LATCH
1	X	X	X	X	RESET
X	X	X	1	DOFF	RESET
X	1	X	1	Set or Reset	RESET
0	X	0	0	Set or Reset	SET
0	0	X	1	Set or Reset	SET

The input signals are processed by the MDOE formatter and the set and reset outputs are input to SR latch 1472. The output of SR latch 1472 is input to AND gate 1474. The second input to AND gate 1474 is the HMEOE* signal on line 1517 which is input to the negative-true input of the gate. The output of the gate is the MDOE signal on line 1503.

In the operation of the circuits in FIGS. 37 and 38, the SR latch set and reset edges, viz., the ERES and ESRT signals, must not overlap the driver off edge, viz., the DOFF signal. Further, AND gates 1464 and 1474 provide a means for overriding the output of latches 1462 and 1472, respectively, by permitting the respective output enable (HDOE or MDOE) to be gated with the HMEOE* signal.

FIG. 39 shows data formatter 1444. The signals input to the data formatter were described in discussing FIG. 36. Formatter 1444 includes data formatter control circuitry 1480, SR latch 1482, and exclusive-OR gate 1484. This control circuitry operates according to the truth table in Table 14.

TABLE 14

FORMAT<1..0>	FOMDATA	Set EDGE	Reset EDGE	Invert
RC=00	0	PASS	PASS	1
	1	PASS	PASS	0
RZ=01	0	—	PASS	0
	1	PASS	PASS	0
R1=10	0	PASS	PASS	0
	1	PASS	—	0
DNRZ=11	0	—	PASS	0
	1	PASS	—	0

The INVERT output of data formatter control 1480 is the first input to exclusive OR gate 1484. The input signals are processed according to the logic values. The set and reset outputs are input to the SR latch 1402. The output of the SR latch is the second input to exclusive-OR gate 1484. The output of the gate is the FMTOUT signal on line 1507. Exclusive-OR gate 1484 is used to invert the data during "return to complement" formatting ("RC"), as will be explained.

Referring now to FIG. 40, data may be transmitted using one of four formats. These formats where shown in Table 14. They are return to complement ("RC") format at 1529, return to zero ("RZ") format at 1533, return to one ("R1") format at 1531, or delay no return to zero ("DNRZ") format at 1535. In the figure, a rising edge indicates a set timing edge, and a falling edge indicates a reset timing edge. These four data formats permit the use of complex patterns while still using only one data bit per pin.

Table 14 shows the function of each format. The RC format permits all data to pass to SR latch 1482, regardless of its value, but inverts the latch output. The RZ format passes all data with a logic high value, but only the reset timing edge is passed with a logic low value. The R1 format passes all data with a logic low value, but only the set timing edge is passed with a logic high value. In the DNRZ format, the only logic low value that will be passed is the reset timing edge when it has a true logic state; and similarly, the only logic high value that will be passed is set timing edge when it has a true logic state. The SR latch set and reset timing edges cannot overlap or operation may be unpredictable.

FIG. 41 shows soft-drive output enable formatter 1446. The formatter consists of two sections. The first is a series of four NAND gates with an inverter at the output of each NAND gate, and the second is a series of NOR gates with an inverter at the output of each NOR gate. The first section includes NAND gates 1559, 1563, 1567, and 1571. Each is a three input device with the LCYC* signal on line 1495 and the FMTOUT signal on line 1507 at two of the inputs. The third input to the NAND gates is one of the bits of the 4-bit SDHEN signal on line 1491. Under normal operations, the logic value of the 4-bits of the SDHEN signal determine the value of the outputs of the NAND gates, assuming the gates are enabled.

The outputs of NAND gates 1559, 1563, 1567, and 1571 are processed by inverters 1561, 1565, 1569, and 1573, respectively. The outputs of the inverters form the 4-bit SPG* signal on line 1497.

The second section is similar to the first, except for the gates and the logic value of the LCYC* signal. The second section has four NOR gates with an inverter at the output of each. The second includes NOR gates 1575, 1579, 1581, and 1585. Each is a three input device with the LCYC signal on line 1593 and the FMTOUT signal on line 1507 at two of the inputs. The third input to the NOR gates is one of the bits of the 4-bit SDLEN signal on line 1493. Under normal circumstances, the logic value of the 4-bits of the SDLEN signal determine which output of the NOR gates will be asserted, assuming the gates are enabled.

The outputs of NOR gates 1575, 1591, 1581, and 1585 are processed by inverters 1577, 1579, 1583, and 1589, respectively. The outputs of the inverters form the 4-bit SNG signal on line 1499.

Soft driver 1452 preferably is enabled during the last cycle of a pattern sequence. Soft-drive output enable formatter 1446 uses the LCYC* signal to gate on and off the high and low soft driver enables, the 4-bit SDHEN signal, and the 4-bit SDLEN* signal. Soft driver 1452 includes separate high and low soft driver enables so that different levels of current may be programmed for driving a logic high and a logic low value. The HMEOE* signal can be used to gate the soft drive output enables off.

FIG. 42, is a block diagram of sampling circuitry 1378. This circuitry is used to sample the output signals from HME pins. Sampling circuitry 1378, preferably includes 5-state sampler 1500, 2-state timing sampler 1502, and short sensor 1504. 5-state sampler 1500 includes analog circuitry for determining whether an HME output signal is a driven high or low, non-driving high or non-driving low, or an unknown level. Before 5-state sampler 1500 is disclosed in detail, the 2-state sampler 1502, short sensor 1504, and the analog support circuits will be disclosed.

Again referring to FIG. 42, 2-state timing sampler 1502 preferably is a flip-flop that is clocked on the positive edge of the clock signal. The output of the flip-flop is TTL timing sample signal on line 1601. The input to flip-flop 1502 is the HME bus signal on line 1304 and the signal that clocks the flip-flop is the SAMPLE signal on line 1593. The 2-state sampler 1502 is controlled by strobing the SAMPLE signal.

FIG. 43 shows a detailed diagram of short sensor 1504. The short sensor circuitry at 1504 and OR gate 1506 (FIG. 42) combine to provide the SHORT_OUT signal on line 1599.

Generally, short sensor 1504 is used to detect when a hard-driver has been continuously shorted for more than about 200 ns. Sensor 1504 includes two identical resistor-capacitor circuits 1510 and 1512 that are used for detecting short circuits. Circuit 1510 detects shorts when the data channel is driving a logic high value for the FOMDATA signal and when sensing a logic low value for the HME* signal. Circuit 1512 detects shorts when the data channel is driving a logic low value for the FOMDATA* signal and when sensing a logic high value for the HME signal.

Circuits 1510 and 1512 will now be described. Taking first circuit 1510, the input signals to NAND gate 1631 are the HME* signal on line 1649, the FOMDATA signal on line 1415, and the HDOE signal on line 1501. The output of NAND gate is input to the gates of field effect transistors ("FETs") 1637 and 1633.

Circuit 1512 is the same as circuit 1510 except for the input signals. The inputs to NAND gate 1635 are FOMDATA* signal on line 1651 and the HME signal on line 1304. The HDOE signal on line 1501 is the same. The circuit that includes NAND gate 1635, FETs 1639 and 1641, resistor 1524 and capacitor 1522 is same as the corresponding elements in circuit 1510. The description of circuit 1510 applies here, and is incorporated by reference.

In operation, capacitors 1518 and 1522 charge slowly through resistor 1520 and 1524 when a conflict is being detected, and discharge quickly through FET 1635 and 1641 when there is no conflict. When the voltage on either of the capacitors charges to a logic high value, the bit is determined to be shorted, and the SHORTED error signal bit is latched in SR latch 1516 through OR gate 1514. A RESET signal is used to clear the SHORTED error signal.

Referring to FIG. 42, OR gate 1506 is used to propagate a chip-wide short-sensor status in a daisy-chain fashion. That is, the SHORT_OUT signal on line 1599 of the first data channel 1350 becomes the SHORT_IN signal to OR gate 1506 of the next data channel 1350.

In operation, the HMS drives signals to the HME pins, and then samples the HME pins to determine the HME's response. When an HME pin is known to be a dedicated input pin, the CIC can hard drive the pin without concern. When an HME pin is known to be a dedicated output pin, the CIC need only sample the pin. In these cases, the CIC need not determine the status of the pin prior to driving or sensing it.

The driving and sensing of I/O pins is not as simple. The CIC must not interfere with the sensing of an output and must not force pins to the wrong state. The CIC of the present invention uses soft drivers to drive I/O pins. The soft-drivers drive an I/O pin in such a manner that the pin is driven when in the input state, yet if the pin is itself driving, the CIC allows the HME to overdrive the CIC. The driving and sensing of all pins will now be discussed.

FIG. 44 shows the three main analog sections of CIC 1300. These sections are current references 1554, 1556, and 1558, HME pin drivers 1550, and 5-state sampler 1500. Each of the data channels 1350 includes its own driver 1550 and sampler 1500.

Current references 1554, 1556, and 1558 generate control voltages for current sources in driver 1550. Driver 1550 uses the current sources as pin drivers to reduce current spikes during conflicts between the HME and CIC 1300 (using the medium-drivers), and

also to perform unobtrusive 5-state sampling of the HME outputs (using the soft drivers).

The respective input to current reference 1554, 1556, and 1558 is ISN on line 1621, ISP on line 1673, and IM on line 1675. Current references 1554, 1556, and 1558 use these signals to generate control voltages for use by the soft and medium drivers. These are the low soft-driver VSNCL voltage on line 1661, the high soft-driver VSPCL voltage on line 1663, and the medium-driver VMNCL voltage on line 1665 and the medium driver VMPCCL voltage on line 1667.

The ISN, ISP, and IM lines connect to ground through external resistors. Preferably, they are connected through 2.5K, 1.25K, and 2.5k ohm resistors, respectively. It is understood that other resistor values may be used to vary the output of the respective current sources.

As examples, FIGS. 45 and 46 are representative I/V curves based on resistors of different sizes being connected in the IM line 1675.

FIG. 47 shows the HME pin drivers shown generally at 1550 in FIG. 44. The drivers are hard driver 1448, medium driver 1450, and soft driver 1452 (FIG. 36). Medium driver section 1450, and soft driver section 1452 are preferably current limited. The outputs of hard, medium and soft-driver sections are input to the HME pins.

Hard driver 1448 is used to drive an HME pin known to be a dedicated input pin. Hard driver 1570 is preferably not used to drive HME I/O pins except in pulsed mode.

Hard driver 1448 is powered by HMEVCC. The HNG signal on line 1783 and the HPG* signal on line 1701 are the control signals for output enabling the hard driver. These signals are generated by the connected circuit. Since the medium driver uses the same circuit to generate the MPG* signal on line 1711 and the MNG signal on line 1713, the circuit of the hard driver will be described with the corresponding reference numbers for the medium driver circuit shown in parenthesis.

The HDOE (MDOE) signal on line 1501 (1503) is input to NAND gate 1705 (1715) and its complement is input to NOR gate 1707 (1717) after being inverted by inverter 1709 (1719). The second input to both gates is the FMTOUT signal on line 1507. When the formatted data represented by the FMTOUT signal is input to the gates, the HDOE (MDOE) signal will output enable both gates when it is asserted. Accordingly, this will assert the HPG* (MPG*) signal on line 1701 (1711) and the HNG (MNG) signal on line 1703 (1713) depending on the state of the FMTOUT signal.

Medium-driver 1572 is preferably designed to drive HME I/O pins during the presentation of the history sequence. Medium-driver 1572 has switching times on the order of 10 ns, however, because it is a current source, the driver is current limited. This minimizes current spikes which might occur if there is a conflict between the HME and the medium driver. The medium-driver preferably is current limited as shown in FIGS. 45 and 46.

In these figures, the low medium-driver current limit is programmable preferably from 8 mA to 16 mA and the high medium driver current limit is programmable preferably from -4 mA to -8 mA. This corresponds to the resistor in the IM line having a value of 2.5K.

The control signals for medium-driver 1572 are the MNG signal on line 1713, MPG* signal on line 1711, VMPCCL signal on line 1667, and VMNCL signal on

line 1665. The MPG* and MNG signals have been discussed. The VMPCCL and VMNCL signals are control voltages generated by the low and high medium drivers current references.

Referring to FIGS. 48A-48E, soft driver I/V characteristic curves will be described. The low soft drivers of the CIC are capable of non-intrusive driving during sampling. The low soft driver, CIC sensing circuitry, and related CMC software have several features to support this capability. Referring to FIGS. 48A and 48B, which are representative soft-drive low I/V characteristic curves for driving TTL, NMOS, CMOS and LCMOS HME pins, these features will be described.

The low soft driver drives toward the programmable drive voltage, VSL. Preferably, the CMC software running on the CPU sets VSL to an appropriate logic low voltage depending upon the HME logic family.

As shown at Point 1 in FIG. 48A, the low soft driver is current limited. This current limit, ISL, is programmable on a pin-by-pin basis. Preferably, the CMC software running on the CPU constrains the low soft driver current limit such that it is less than the source current of the HME pin in the driving high state. Thus, if the HME pin is in a driving high state, the HME pin will overdrive the enabled CIC low soft driver to a logic high.

Preferably, the CMC software running on the CPU sets the low soft driver current limit, depending upon the VSL used, such that if the HME pin is in a non-driving state, then the CIC low soft driver will drive the HME pin to a logic low. For example, for TTL HME pins, the CMC software constrains the low soft driver current limit such that it is more than twice the input leakage current for the HME pin in the non-driving low state. Referring to Point 2 of FIG. 48A, with VSL set at 0.4 V, the low soft driver will drive a TTL HME pin which is in the non-driving state to 0.6 V, which is a valid logic low.

Preferably, the CMC software running on the CPU sets a reference voltage, VLTH, for comparison with the HME pin voltage, depending upon the VSL used and the low soft driver current limit used, so that if the HME pin is in a driving low state and the CIC low soft driver is enabled, then the HME pin voltage will be less than VLTH. For example, at Point 3 in FIG. 48A, the "reverse" current of the low soft driver is less than ISL, the sink current of the HME pin is greater than ISL, and the I/V characteristic curve in this region is essentially linear. In this case, therefore, the CMC software sets VLTH to be closer to VSL than to 0.0 V.

The high soft drivers of the CIC are capable of non-intrusive driving during sampling. The high soft driver, CIC sensing circuitry, and related CMC software have several features to support this capability. Referring to FIG. 48C, 48D, and 48E, which are representative soft-drive high I/V characteristic curves for driving TTL, NMOS, CMOS and LCMOS HME pins, these features will be described.

The high soft driver drives toward a programmable drive voltage, VSH. Preferably, the CMC software running on the CPU sets VSH to an appropriate logic high voltage depending upon the HME logic family.

As shown at Point 1 in FIG. 48C, the high soft driver is current limited. This current limit, ISH, is programmable on a pin-by-pin basis. Preferably, the CMC software running on the CPU constrains the high soft driver current limit such that it is less than the sink current of the HME pin in the driving low state. Thus,

if the HME pin is in a driving low state, then the HME pin will override the enabled CIC high soft driver to a logic low.

Preferably, the CMC software running on the CPU sets the high soft driver current limit, depending upon the VSH used, such that if the HME pin is in a non-driving state, then the CIC high soft driver will drive the HME pin to a logic high. For example, for TTL HME pins, the CMC software constrains the high soft driver current limit such that it is more than twice the input leakage current for the HME pin in the non-driving high state. Referring to Point 2 of FIG. 48C, with VSH set at 3.0 V, the high soft driver will drive a TTL HME pin which is in the non-driving state to 2.6 V, which is a valid logic high.

Preferably, the CMC software running on the CPU sets a reference voltage, VH_{TH}, for comparison with the HME pin voltage, depending upon the VSH used and the high soft driver current limit used, so that if the HME pin is in a driving high state and the CIC high soft driver is enabled, then the HME pin voltage will be greater than VH_{TH}. For example, for TTL HME pins, at Point 3 in FIG. 48C, the "reverse" current of the high soft driver is less than ISH, the source current of the HME pin is greater than ISH, and the I/V characteristic curve in this region is essentially linear. In this case, therefore, the CMC software sets VH_{TH} to be closer to VSH than to the maximum HME drive voltage.

Table 15 shows the preferred settings for the programmable voltage levels used in driving and sampling HME pins of the above-indicated logic families.

TABLE 15

FAMILY	VL _{TH}	VSL	VLOGL	VLOGH	VSH	VH _{TH}	HMEVCC
TTL	0.35 V	0.4 V	0.8 V	2.0 V	2.9 V	3.05 V	5.0 V
NMOS	0.25	0.4	0.8	2.0	2.9	3.2	5.0
CMOS	0.25	0.4	1.0	3.5	4.5	4.7	5.0
LCMOS	0.25	0.4	0.6	2.5	2.8	3.0	3.3

Similarly, Table 16 shows the preferred settings for programmable current levels used in driving and sampling the HME pins.

TABLE 16

FAMILY	ISL	ISH
TTL	2.0 mA	-0.5 mA
NMOS	0.5	-0.5
CMOS	0.5	-0.5
LCMOS	0.5	-0.5

The preferred embodiment of 5-state sampler 1500 is shown in FIG. 49. FIG. 50 also will be referred to in disclosing the 5-state sampler. The 5-state sampler includes analog voltage comparators 1600, 1602, 1604, and 1606. The comparators compare the voltage sensed at HME pins with the four reference voltages. These are VLOGH on line 1175, VLOGL on line 1173, VH_{TH} on line 1177, and VL_{TH} on line 1181. The outputs from the comparators are input to two NOR gates.

The inputs to NOR gate 1631 are the outputs of comparators 1600 and 1602. The inputs NOR gate 1635 is either the output of comparators 1604 or 1606, and the output of exclusive-OR gate 1633. The inputs to the exclusive-OR gate are the FOMDATA signal on line 1415 and the HME signal on line 1304.

Data selector 1608 selects whether the output of comparator 1604 or 1606 will be the input to NOR gate

1635. The FOMDATA signal is used to determine what is being soft driven to the HME pin.

The output of the comparators and the logic circuitry is latched by registers 1610, 1612, and 1614. The output of comparator 1600 is input to the data input of register 1610, the output of NOR gate 1631 is input to the data input of register 1613, and the output of NOR gate 1635 is input to the data input of register 1614. The three registers are clocked by the SAMPLE* signal which is the SAMPLE signal on line 1593 that has been inverted by inverter 1508. The SAMPLE signal may be asserted after a sufficient time delay to permit settling of the signal to occur. The output of registers 1610 is the VALUE SAMPLE signal on line 1611, the output of register 1612 is the U SAMPLE signal on line 1613, and the output of register 1614 is the HIZ SAMPLE signal on line 1615. This sampling operation may be performed on one or more pins while one or more different HME pins are being driven. Because each data channel includes its own sensing and driving circuitry, multiple pins of the HME are sensed simultaneously. The results of sampling are returned to the simulator.

There are three bits of sampled information output for the five state sampling circuitry. However, they provide five states of useful information: driving high ("H1"), non-driving high ("Z1"), driving low ("H0"), non-driving low ("Z0"), and unknown ("U"). An output of high or low (HIZ_SAMPLE deasserted), indicates that the HME pin is an output pin and that it is driving a logic high or logic low value. An output of Z1 or Z0, (HIZ_SAMPLE asserted) indicates that the

HME pin is in a non-driving state. An output of U, which is asserted as the U SAMPLE signal, indicates that the HME pin was sampled at an unknown level (between VLOGL and VLOGH).

FIG. 50 shows the relationship among the supply voltages, voltage references, and HME pin voltages shown in FIG. 49. The voltages shown on FIG. 49 are the preferred voltages for use in testing TTL logic circuits. The values of the reference voltages will differ depending upon the technology of the device being tested (e.g., TTL, ECL, CMOS, NMOS and LCMOS). These values may be selected through the software shell. The VSH and VSL voltages shown in FIG. 50 are the voltages at which the CIC soft drives a logic high and a logic low value, respectively.

Table 17 discloses how determination are made regarding the status of sampled HME pins:

TABLE 17

1. VH_{TH} < HME pin voltage ≤ HMEVCC. The HME pin voltage is above both VSL and VSH, which are the target voltages of the soft drivers, therefore, the HME must be in the H1 state. If the soft drivers are also driving a logic high value, the measured voltage is merely driven closer to HMEVCC.
2. VLOGH < HME pin voltage < VH_{TH}. If the soft drivers are driving a logic high value, the HME pin is in the Z1 state (the HME is an input pin and the soft drivers are driving at VSH). If the soft drivers are

TABLE 17-continued

	not driving, the HME is an output pin, and it is in the HI state. The logic circuitry of 5-state sampler 1500 determines whether or not the soft drivers are driving.
3.	VLOGL < HME pin voltage < VLOGH. The HME pin voltage is between a valid logic low value and a valid logic high value, and therefore is classified as unknown. This state indicates that a problem exists.
4.	VLTH < HME pin voltage < VLOGL. If the soft drivers are driving a logic low value, then the HME pin is in the ZO state (the HME is an input pin and the soft drivers are driving at VSL). If the soft drivers are not driving, the HME is an output pin, and it is in the HO state. The logic circuitry of 5-state sampler 1500 determines whether or not the soft drivers are driving.
5.	HME pin voltage < VLTH. The HME pin voltage is below both VSL and VSH, so the HME is in the HO state.

This sampling method permits the HME pin to dominate whenever both the HME pin and the soft drivers are driving (i.e., when the HMS is trying to soft drive an I/O pin which is acting as an output). It also permits the user to know which of the high impedance zones (i.e., Z1 or Z0) the HME pin is in. For example, if the HME pin was an output pin driving a logic high value and then becomes a high impedance input, the CIC will soft drive low during the next calculation.

Referring again to FIG. 32, status/read control 1372 of data channel 1350 is a 16:1 multiplexer which used primarily for diagnostic purposes and for reading sampled results. The multiplexer receives as inputs the data described in the Table 18 and outputs data to the Pattern Bus. Control 1372 is read by first strobing the 4-bit address on the four pattern lines during CPU access mode and latching the address with the AS* signal. Then the RD* signal is asserted. The signals set forth in Table 18 provide the status of the individual data channels, with the exception of the parity status and RESET signals, which give the entire status of CIC 1300.

TABLE 18

Ad- dress	Source	Source Description
0	FMT OUT	SR Data Formatted output.
1	MDOE	SR Medium-Drive output enable.
2	HDOE	SR Hard-Drive output enable.
3	CTL_OUT	Control Register CTL_OUT bit.
4	LCYC	Delayed version of SEQEND.
5	FORMATTER HMDEN*	Formatter hard/medium drive enable output from the Control Register.
6	FOMDATA	Formatter data output from the Control Register.
7	ERROR DATA	Last input pattern data before ERROR* is asserted.
8	SHORTED	Short sensor status
9	U SAMPLE	5-state U sample
10	HIZ SAMPLE	5-state Z sample
11	VALUE SAMPLE	5-state 0/1 sample
12	TTL Timing Sample	2-state timing measurement sample.
13	Unused	
14	Parity Status Register	PAT<0> contains the input PARITY bit. PAT<1> contains the output parity status. A value of 1 for this bit indicates a parity error.
15	RESET	Reading from this address causes the internal RESET to be asserted.

TABLE 18-continued

Ad- dress	Source	Source Description
5		Also, the value read is the CIC version number.

Device Adapter Board

10 The primary function of DAB 268 is to fixture an HME to the PEL of an HMS. FIG. 52 shows a top view of a representative DAB. It is an 80 pin DAB. It is understood that the DAB may be a 160, 240, or 320-pin DAB, or be a DAB with more than 320 pins, and still be in the scope of the present invention.

15 Two main components of the DAB 268 shown in FIG. 52 are system connector 1702, which connects the DAB to the PEL of the HMS and, pad array 1704, referred to as the HME "footprint," to which an HME or an HME socket connects. The footprint is designed to permit a family of HMEs that have different pin-out configurations to be fixtured on the same pad array, without requiring any custom wiring on the DAB. DAB 268 also includes an array of power and ground pads, which may be connected directly to the pads that are intended to connect to HME power and ground pins.

20 System connector 1702 is used for communicating between the HME and the PEL. There may be one or more system connectors on a given DAB depending on the number of signal pins the DAB is intended to support.

25 The HME footprint preferably supports a class of HME package types having up to a predetermined number of device pins. The footprint pads preferably are laid out in a matrix with 0.1 inch centers. For example, the footprint for supporting a family of device packages with pin grid array ("PGA") pin-out configurations is a square matrix. The footprint for a family of device packages with dual-in line package ("DIP") pin-out configurations is a rectangular matrix with multiple parallel rows on 0.1 inch centers. Another example is the footprint for a family of device packages with plastic leaded chip carrier ("PLCC") pin-out configurations or quad-flat pack ("QFP") pin out configurations is a rectangular matrix also with multiplex parallel rows 0.1 inch centers. The footprint, therefore, may be configured in any manner which supports the direct connection of HMEs to the DAB. It is to be understood that custom DABs may be constructed for any special HME requirements, such as 50-mil PGA or high-density surface-mount devices.

30 Support for a class of HME package types is achieved by designing the HME footprint to be a superset of a specific class of HME package types. For illustration purposes only, the footprint of the 80-pin PGA DAB shown in FIG. 51 will be described.

35 The DAB matrix which supports the 80-pin PGA package has a total of 169 pads arranged in a square 13-by-13 matrix, on 0.1 inch centers in both dimensions. It is to be understood that the total number of signal pads of a given DAB footprint may exceed the number of signals available from the system connector. For example, the above-mentioned DAB footprint has 169 pads to which a maximum of 80 pins are connected. In connecting the footprint pads to the system connector pins, the 80 pins from the system connector are initially routed to one pad in the HME footprint. Each pin may

then have one or more additional matrix pads connected to it. The pins that connect to more than one pad preferably are routed in a manner to take advantage of the concentric ringed nature of standard PGA packages. FIGS. 52A-52F illustrate the package pin-out configuration supported by the 80 pin PGA DAB footprint. If a device has a pin-out configuration that is a subset of any of the arrays that are shown in these figures, it can be supported by the DAB.

The impedances of the traces that connect the system connector pins to the HME footprint pads are controlled. Such impedances are selected to match the combined impedance of the PEL drivers and any series dampening resistors. Impedance matching decreases ringing and undershoot of the HME signal lines. Equalizing trace lengths to each of the pads permits very precise timing measurements to be made.

In a preferred embodiment, power supply, HMEVCC, pads and ground pads are distributed throughout the HME footprint for simplifying their connection to the HME footprint pads. These pads preferably are coded by shape: square pads for ground, as shown at 1708, and diamond pads for power, as shown at 1710. These shapes are easily distinguished from the round pads which are used for all other signals.

To accommodate mounting an HME to the DAB, vias are provided for every HME footprint pad. The vias are channels for the traces and allow the connection of the footprint pad to the system connector pins. Further, the HMEVCC and ground connections are on the front side of the DAB to allow the easy connection to the footprint pads.

In the preferred embodiment, DAB 268 includes circuitry to permit the DAB to be inserted or withdrawn from a powered HMS. This is referred to as a live insertion capability. Live insertion eliminates the need to power-down the HMS to insert or remove a DAB. This is particularly useful in systems supporting multiple users. The live insertion capability can be used with or without a specially designed HME footprint.

Live insertion is preferably accomplished in three phases: dissipation of electrostatic charge, equalization of signal ground potential, and gradual powering of the DAB power supplies. Each phase will now be discussed.

The system connector is designed such that its pins mate to the PEL connector at three different times. The first pin of the DAB system connector to mate with the female connector of the PEL connects to the PEL (and HMS) chassis ground. This dissipates any stored electrostatic charge on the DAB. The DAB system connector pin that makes the first contact is 0.5 inches longer than the normal connector pins.

The second pin of the DAB system connector to mate with the female connector of the PEL is associated with signal ground. When it mates it allows the equalization of the DAB and PEL signal grounds. If the signal grounds were not equalized before the main signal pins connect, it could result in a damaging voltage being sent to the PEL (and HMS). The DAB pin that makes this second contact is approximately 0.25 inches longer than the normal connector pins. After this mating, the main body of normal connector pins mate.

An important characteristic of the live insertion circuitry is the slow ramp-on of the power supplies. This substantially eliminates the large in-rush of current which might otherwise occur as the capacitors on the DAB and the HME charge when the power supplies

are connected to the DAB. Unless the current in-rush were eliminated, it could cause damage to the HMS, HME, and DAB. This instantaneous in-rush of current could also cause voltage transients on PEL 266.

To accommodate a wide variety of circuits and support circuits that require voltages other than +5 V, the DAB is provided with three additional power supplies. These are the -12 V at 1712, -5.2 V at 1714, and the +12 V at 1716.

The DAB of the present invention also provides a means to regulate the HMEVCC voltage to accommodate HMEs with different supply voltages. This is effected by the circuit shown in FIG. 53. In the circuit, the +5 V supply voltage is connected to the input of voltage regulator 1773 and the terminal 1783 of switch 1779. The output of the voltage regulator is determined by the value of the resistor 1724 and 1725. Table 19 shows the relationship of the resistors and a number of output voltages.

TABLE 19

VREG	RA value in ohms	RB value in ohms
3.5 V	100	180
3.3 V	91	150
3.2 V	43	68
3.0 V	47	68
2.9 V	91	120
2.8 V	120	150
2.5 V	91	91
2.4 V	100	91
2.2 V	120	91
2.0 V	91	56
1.8 V	75	33
1.5 V	120	27

As shown in FIG. 53 either the regulated voltage at 1787 or the +5 V signal at 1783 may be selected for input to the HME after passing through fuse 1789.

It is desirable to connect all power supply pins of the system connector to the female connector of the PEL at the same time to prevent the need for power supply sequencing circuitry. However, when the DAB is being connected to the female connector of the PEL, pins near the edges can be connected and disconnected as the system connector is being rocked into position. If power supply pins are near the edges, it is possible for one power supply to be fully active and for another to be completely shut off as the connection is being made. To reduce the likelihood of this occurring, and, thereby, obviating the need for power sequencing, supply voltage pins are preferably placed near to the center of the system connector.

A public HME must be initializable to a consistent state before each evaluation. Certain HME circuits, for example, divide-by-N counters, cannot be initialized by assertion of a RESET signal. If the HME contains such a circuit, the HME may be initialized by repeatedly presenting a sequence of stimuli which causes the HME to advance to a particular internal state. One or more feedback pads such as FEEDBACK pad 1730 are provided on DAB 268 to allow the HME to transmit signals to the HMS regarding its internal state so it can be determined when the HME is initialized.

In a preferred embodiment of the invention, DAB 268 includes a configuration area. The configuration areas for the 80-pin DAB in FIG. 51 are shown at 1736, 1738, and 1740. Every trace from the system connector passes through one of the configuration areas. Each trace goes through a via from the system connector to

one of the configuration areas, surfaces at the configuration area, and then through another via to a footprint pad. The vias and surface traces are arranged in an orderly pattern, and preferably are numbered.

FIG. 54A shows three representation traces 1801, 1803, and 1807 in a configuration area. The traces may be cut in the configuration to increase the versatility of DAB 268 or the HME footprint. For example, the surface trace may be cut to permit the use of a power supply outside the range of the PEL output protection circuitry. That is, if the PEL is only designed to withstand continuous short circuits to HMEVCC or ground, the corresponding trace must be cut when the HME pin is connected to a voltage higher than HMEVCC. Surface traces may also be cut to connect a level translator (e.g., TTL-ECL or ECL-TTL) in series between the system connector and the HME footprint. FIG. 54B shows a representative trace 1807 that may have been cut for one of the above described purposes.

A DAB can only support an HME having a pin count which does not exceed the number of HME signal lines on the system connectors. This limit can be exceeded by cutting the trace to an HMEVCC, ground, or no-connect pin, in the configuration area, and connecting the system connector-side via of the cut trace to a previously unconnected HME signal pin.

DAB 268 preferably includes one or more work areas such as show at 1760. Work area 1760 include arrays of through-hole pads which are not connected to any signals or power supplies. Work area 1760 preferably is surrounded by alternating HMEVCC and ground pads for use in powering the circuitry disposed in the work area. The work area may be used to implement circuitry to support the HME. For example, special feedback logic for decoding a particular HME state into a single rising or falling edge may be mounted on a work area. Other circuits, such as level translators and analog support circuitry for the HME, including special HME clock drivers, indicators, and other HME support logic, may be mounted on the work area.

Many HMEs which specify a minimum operating clock frequency contain a substrate bias generator or other such circuitry in which operation is sustained by the presence of a clock. If the clock stops, or if the clock frequency becomes too low, such devices may require a very long time warm-up before use, or may even sustain permanent damage. To prevent this, keep alive circuitry preferably is provided on DAB 268 to ensure that the HME is clocked at all times. This circuitry preferably includes a register which is output enabled when ever the HME is not in use. This output is the register is connected to the HME clock pins. A range of clock frequencies may be provided to accommodate a predetermined range of HME operating frequencies.

The DAB has two indicators to provide visual indication of the states of the DAB. The first is LED 1770. This LED, when lit, indicates that HMEVCC is available on the DAB. This state is reached once the DAB is connected to the PEL and HMEVCC has ramped up. The second is LED 1772. This is the in-use LED. This LED turns-on when the first instance of the HME is created and goes off when the last instance of the device is released.

The DAB also has two test points that can be used for synchronizing, for example, the operation of external test equipment with that of the HME. The first test point is at 1776. This is used primarily by the software associated with the HME. Two representative events

are marked by the changes in the state of the signal output from the first test point, viz., the beginning and end of the evaluation period. The second test point is a pulsed output that may be used to mark specific events during the period the HME is being evaluated. Either of the two test points may be connected, for example, to an external device for synchronizing its operation with the operation of the HME.

Analog sense connector 1780 is jumpered to the analog outputs of the HME. Monitoring of the analog output provides information regarding analog changes occurring in the HME during evaluation that would not be revealed if only the digital signal outputs were monitored.

Operation

Hardware modeling system 100 uses a five step method to evaluate a public HME. These five steps are: programming CIC 1300 and other hardware, resetting the HME to a known state, restoring the HME to the last measured state, stimulating the HME with a new stimulus pattern, and measuring the results of the stimulation.

The programming step preferably is accomplished by sending several patterns of programming data to PEL 266 in the initial patterns of the history sequence during the pattern presentation mode. According to the present invention, preferably 22 patterns of programming data are sent.

The resetting step is accomplished by clocking a RESET pattern sequence to the HME, if the HME is the type that will accept and act upon it. If the HME is not resettable in this manner, feedback looping is performed within the VRAM shift registers to bring the HME to a known state. The HME must be returned to a known state before the history sequence is presented, otherwise, there is no way to ensure that the subsequent restoring step will restore the HME to the last measured state.

During the restoring step, patterns are presented to the HME to restore it to its last measured state. The entire sequence of patterns produced by the HMS from the time simulation began until just prior to the current event is presented to the HME. By completely replaying the pattern sequence before each new stimulation, the simulator may take whatever time is needed to perform analyses of the simulated external system. Without the restoring step, dynamic devices, which have minimum clock frequency specifications, would decay to unknown states between simulator accesses. The restoring step also permits the HME to be shared among more than one simulator and permits a single HME to represent a replicated device within the system being simulated. The restoring step brings the HME to a state from which timing measurements and other measurements may be performed. During the restoring step, patterns are sent to the CIC generally as one bit per pin to reduce the required pattern storage. Output pins of the HME are left floating during the restore step.

The single bit per pin of the pattern specifies a driving direction for the HME pin driver. Strict input pins of the HME are hard driven in the specified direction. I/O pins are driven using the CIC current-limited medium driver as follows: if the HME pin is driving, the CIC medium driver drives in the same direction as the HME pin, otherwise, the CIC medium driver drives in the direction that the simulated external network (connected to the HME pin in the simulated design) is driv-

ing. Strict output pins are effectively not driven during the restoring step.

During the stimulating and measuring steps, new stimulus is presented to the HME pins, and the outputs are sampled. In this process, the HMS waits a predetermined interval for the device outputs to settle, and then samples all outputs simultaneously. The changes are returned to the simulator. Scheduling delays are also returned with the changes. The delays can be obtained from the delay table specified in the shell software. This method allows the use of worst-case timing values or ranges from the device data sheet.

The system of the present invention can perform its own timing delay measurements, such as determining the output delay of each HME output transition. This allows the HMS to create its own Timing file. The SAMPLE timing edge on line 1593 and register 1502 (FIG. 42) are used to perform the binary search shown in FIG. 55, until the output delay is determined. Binary searching consists of continually presenting the same pattern sequence to the HME, and placing the SAMPLE edge at different times during the measurement stage to determine when each HME output pin changes state. As shown, the rising SAMPLE edge at 1852 is placed at time A, and the HME pin value 1850 is sampled as a logic high value. The same pattern sequence is again presented to the HME, and SAMPLE edge 1852 is placed at time B. The output data is sampled as a logic low value. The measurement is continually iterated by repeating the pattern sequence, and continually cutting the interval (A to B) in half. If the sampled value is a logic low value, the next sampling time is increased. If the sampled value is a logic high value, the next sampling time is decreased. The final result of the process when the interval size reaches a small predetermined value is the transition of the signal.

An HME includes several types of pin connections, and each requires a special driving or sampling technique. According to the present invention power, ground, and "no-connect" pins are not driven. Dedicated input pins are driven by the hard drivers, which are turned on and remain on throughout all of the stages of a pattern presentation. Dedicated output pins are never driven with either the hard or medium drivers. During the restoring and stimulating steps of pattern presentation, the output pins are left floating. During the measurement step, the output pins are soft-driven to determine whether the output is driving high or driving low, or non-driving high or non-driving low or unknown. HME I/O pins operate as inputs sometimes and as outputs at other times. HME I/O pins are preferably medium-driven during the restoring and stimulating steps. I/O pins may also be driven by pulsing the hard drivers, preferably, using 15 ns pulses. After these hard driver pulses, the soft drivers may continue to drive in the same direction. Using pulsed hard drivers causes faster input voltage transitions than using medium drivers. This mode of use can be enabled by the HMS user for fastest pattern presentation rates.

HME input and I/O pins may be further classified into several pin types, relating to the pin function. These classifications include data pins, eval pins, store pins, edge_fall pins, edge_rise pins, and I/O store pins. A data pin is a pin which affects neither the output state nor the internal state of an HME. An eval pin is one which may affect the outputs of an HME, but does not affect the HME's internal state. A store pin may affect the output and/or the internal state of the HME.

Edge_fall and edge_rise pins are store pins having only one edge which can cause internal state changes. The edge_fall and edge_rise pins affect the output and state of the HME only on the falling and rising edges, respectively. An I/O store pin is an I/O pin which is also a store pin. Patterns preferably are presented to this type of pin during the stimulating step using two bits per pin.

During the stimulating step, patterns are sent to the HME according to the previously described method using one bit per pin, but patterns may be sent as two bits per pin if an I/O store pin is changing from input to output or output to input. FIG. 56 shows an example of an I/O store pin. The clock of register 1900 within the HME is connected to an I/O store pin 1902 and the output of an internal tri-state buffer 1904. During the stimulating step, it is not known whether pin 1902 or buffer 1904 is driving the clock input at register 1900.

FIG. 57 shows patterns being presented to an I/O store pin. The patterns are sent to the pin as one bit per pin for all pattern cycles except cycles 1910 and 1912. During these two cycles, the HME pin direction was found to change (during a previous measurement step). During 1910, the HME I/O store pin transitions from non-driving low to driving high. During 1912, the HME I/O store pin transitions from driving high back to non-driving low. During each of these cycles, the appropriate pattern is sent using two bits per pin for every pin of the HME. The first bit is latched within the CIC channel and used as a control bit; it specifies whether the appropriate CIC medium driver will be enabled. The second bit is used for the data value. Thus, at 1910, the two bits for this particular HME I/O store pin would disable the CIC medium driver. The data value is irrelevant. At 1912, the two bits for this I/O store pin would re-enable the CIC medium driver and also begin to driving low.

For private HMEs, only a single stimulus pattern is presented to the HME during evaluation. However, as stated, a public HME is restored to its last known state by presenting a history sequence to the HME. To compress the history sequence, only certain measurements cause the sequence to grow. When a store pin, edge_fall pin, or edge_rise pin, transitions, the history sequence must grow to add one additional pattern as shown in FIG. 58. Because such transitions can change the internal state of the HME, the sequence must grow to ensure that the HME is properly restored by the sequence.

Data and eval pin transitions do not cause the sequence to grow. The current values of the data pins are queued on the host computer system until an eval or store pin changes. When measuring the effects of an eval or store pin transition, the pattern at the end of the history sequence is altered, and the HME is measured again, but the history sequence does not grown.

The following procedure is initiated by CPU 240 to begin a pattern presentation. CPU 240 programs the TG pattern clock frequency, and programs the edge and sampling times. The CPU then allocates the pattern memory in PAMs and sets up the pattern sequence to be played, including PEL control patterns and any device initialization sequence. Note that the pattern sequence may already be set up from a previous evaluation. CPU 240 then sends a start signal to TG 248 for the lanes in which the HME resides. Multiple lanes can be started simultaneously.

The above sequence of events results in the following: TG 248 starts PACs 258 in the selected lanes. PACs

258 sequence through the attached PAMs as programmed, delivering the history sequence patterns to the appropriate PELs. PELs 266 deliver the patterns to the pins of the selected HME via DAB 268. CICs on PELs 266 sample and latch the HME pin states. CPU 240 retrieves the sampled data from PELs 266 and returns it to simulator 164.

Having illustrated and described the principles of our invention with reference to one preferred embodiment, it should be apparent to those persons skilled in the art

that such invention may be modified in arrangement and detail without departing from such principles.

The terms and expressions used herein are terms of expression and not of limitation, and there is no intention, in the use of such terms and expressions, of excluding any equivalents of the features shown and described, or portions thereof, it being recognized that various modifications are possible within the scope of the invention as claimed.

APPENDIXES I, II and III

APPENDIX I

Table of Contents

MC68020	Tab 1
PGA	Tab 2
PGA160	Tab 3

Copyright 1989 Logic Modeling Systems		FILE MC68020A.MDL	DATE 5/25/89	PAGE # 1/2
LINE #	TEXT			
1	{ }			
2	{ * Copyright (c) 1988-1989 by Logic Modeling Systems, Inc. }			
3	{ * All rights reserved. }			
4	{ }			
5	{ * Logic Model main MODEL file for Motorola MC68020 }			
6	{ * 32-Bit Microprocessor }			
7	{ }			
8	device_name MC68020			
9	{ }			
10	{ model revision A,B }			
11	{ shell revision B }			
12	{ }			
13	physical 'MC68020.DEV'			
14	timing 'MC68020A.TMG' { 12.5MHz }			
15	package 'PCA.PIC' { 114 }			
16	adapter 'PCA160.ADP'			
17	options 'MC68020.OPT'			
18	names 'MC68020.NAM'			
19	{ }			
20	{ }			

Copyright 1989 Logic Modeling Systems		FILE MC68020.DEV	DATE 5/25/89 TIME 12:50:07 pm	PAGE # 1/1
LINE #	TEXT			
1	{			
2	* Copyright (c) 1988 by Logic Modeling Systems, Inc. All rights reserved. *			
3	{			
4	* Logic Model DEVICE file for Motorola MC68020 *			
5	{			
6	* 32 Bit Microprocessor *			
7	{			
8	* Data obtained from M68000 Family Reference 1988 *			
9	{			
10	* model_revision A,B *			
11	{			
12	* shell_revision B *			
13	{			
14	technology NMOS			
15	{			
16	device_speed 8 MHz - 12.5 MHz { MC68020RC12, MC68020RC16 }			
17	{			
18	device_setup_time 15 { parameter 48, 12.5MHz }			
19	{			
20	device_hold_time 10 { parameter 47B, 12.5MHz }			
21	{			
22	{			
23	in_pin			
24	{			
25	CLK = C2 : store, keepalive { clock - 10MHz keepalive }			
26	{			
27	CDIS = E1 { cache disable }			
28	{			
29	AVEC = E2 { autovector }			
30	{			
31	BR = E3 { bus request }			
32	{			
33	BGACK = A1 { bus grant acknowledge }			
34	{			
35	BERR = J2 { bus error }			
36	{			
37	TPR[2:0] = M12,J13,J12 { interrupt priority level }			
38	{			
39	DSACK[1:0] = J1,E3 { data xfer/size acknowledge }			
40	{			
41	{			
42	out_pin			
43	{			
44	A[31:0] = A3, B4, C5, A4, B5, A5, C6, B6, { address bus }			
45	{			
46	A6, A7, C7, E7, A8, B8, C8, C9,			
47	B10, A11, B11, C10, A12, B12, C11, A13,			
48	C12, B13, C13, D12, D13, E12, A2, C4			
49	{			
50	SIZ[1:0] = C2,F1 { transfer size }			
51	{			
52	FC[2:0] = F2,F3,E1 { function codes }			
53	{			
54	RWC = E2 { read-modify-write cycle }			
55	{			
56	AS = L1 { address strobe }			
57	{			
58	DS = M1 { data strobe }			
59	{			
60	R/W = L2 { read/write }			
61	{			
62	DSACK = G1 { data buffer enable }			
63	{			
64	TPRMD = F13 { interrupt padding }			
65	{			
66	BG = B2 { bus grant }			
67	{			
68	ECS = G1 { external cycle start }			
69	{			
70	OCS = E13 { operand cycle start }			
71	{			
72	{			
73	io_pin			
74	{			
75	D[31:0] = M1, L3, M2, N2, L4, M3, N3, M4,			
76	{			
77	L5, M4, M5, N5, L6, M6, N6, M7,			
78	{			
79	L8, M9, N9, M10, L9, M10, N11, M12,			
80	{			
81	L10, M11, M12, M13, L12, L13, K12, K13			
82	{			
83	RESET = C1 { reset }			
84	{			
85	HALT = K2 { halt }			
86	{			
87	{			
88	power_pin			
89	{			
90	VCC = A9, D3, M8, N8, M13, D1, D2, E3, G11, G13			
91	{			
92	ground_pin			
93	{			
94	GND = A10, B9, C3, F12, L7, L11, N7, K3, G12, M13, J3, K1, B1			
95	{			
96	{			
97	initialize			
98	{			
99	CLK = 520(1.0)			
100	{			
101	RESET = 520(0.0)			
102	{			
103	HALT = 520(0.0)			
104	{			

Copyright 1989 Logic Modeling Systems		FILE MC68020A.TMG	DATE 5/25/89 TIME 12:50:07 pm	PAGE # 1/5
--	--	----------------------	--	---------------

LINE #	TEXT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

Copyright 1989
Logic Modeling Systems

FILE
MC68020.TST

DATE
5/25/89

PAGE #
1/6

TIME
12:50:08 pm

LINE #

TEXT

1start MC68020

2{ { model_revision 8 }

3{ { shell_revision 8 }

4CLK

5-IPL2

6-IPL1

7-IPL0

8-DSACK1

9-DSACK0

10-CDIS

11-BR

12-BGACK

13-BERR

14-AVEC

15D31

16D30

17D29

18D28

19D27

20D26

21D25

22D24

23D23

24D22

25D21

26D20

27D19

28D18

29D17

30D16

31D15

32D14

33D13

34D12

35D11

36D10

37D9

38D8

39D7

40D6

41D5

42D4

43D3

44D2

45D1

46D0

47-RESET

48-WAIT

49STZL

50SIZE0

51R/-W

52FC2

53FC1

54FC0

55A31

56A30

57A29

58A28

59A27

60A26

61A25

62A24

63A23

64A22

65A21

66A20

67A19

68A18

69A17

70A16

71A15

72A14

73A13

74A12

75A11

76A10

77A9

78A8

79A7

80A6

81A5

82A4

83A3

84A2

85A1

86A0

87RMC

88-CCS

89-IPEND

90-ECS

91-DS

92-DEN

93-Bc

94-AS

95ipatterns {

96{

97{

98{

99{

100{

101{

102{

103{

1040

1050

10625

10725

10825

10927

11075

11175

112125

113125

114175

115175

116225

117225

118275

119275

120325

[illegible]

Copyright 1989 Logic Modeling Systems		FILE PGA.PKG	DATE 5/25/89 TIME 12:50:36 pm	PAGE # 1/1
--	--	-----------------	--	---------------

LINE #	TEXT
1
2	* Copyright (c) 1988 by Logic Modeling Systems, Inc. All rights reserved. *
3
4	* Logic Model PACKAGE map file for JEDEC standard PGA packages
5
6	{ map_revision A }
7
8	package_mapping
9	A1 - A1
10	A2 - A2
11	A3 - A3
12	A4 - A4
13	A5 - A5
14	A6 - A6
15	A7 - A7
16	A8 - A8
17	A9 - A9
18	A10 - A10
19	A11 - A11
20	A12 - A12
21	A13 - A13
22	A14 - A14
23	A15 - A15
24	A16 - A16
25	A17 - A17
26	A18 - A18
27	A19 - A19
28	A20 - A20
29	B1 - B1
30	B2 - B2
31	B3 - B3
32	B4 - B4
33	B5 - B5
34	B6 - B6
35	B7 - B7
36	B8 - B8
37	B9 - B9
38	B10 - B10
39	B11 - B11
40	B12 - B12
41	B13 - B13
42	B14 - B14
43	B15 - B15
44	B16 - B16
45	B17 - B17
46	B18 - B18
47	B19 - B19
48	B20 - B20
49	C1 - C1
50	C2 - C2
51	C3 - C3
52	C4 - C4
53	C5 - C5
54	C6 - C6
55	C7 - C7
56	C8 - C8
57	C9 - C9
58	C10 - C10
59	C11 - C11
60	C12 - C12
61	C13 - C13
62	C14 - C14
63	C15 - C15
64	C16 - C16
65	C17 - C17
66	C18 - C18
67	C19 - C19
68	C20 - C20
69	D1 - D1
70	D2 - D2
71	D3 - D3
72	D4 - D4
73	D5 - D5
74	D6 - D6
75	D7 - D7
76	D8 - D8
77	D9 - D9
78	D10 - D10
79	D11 - D11
80	D12 - D12
81	D13 - D13
82	D14 - D14
83	D15 - D15
84	D16 - D16
85	D17 - D17
86	D18 - D18
87	D19 - D19
88	D20 - D20
89	E1 - E1
90	E2 - E2
91	E3 - E3
92	E4 - E4
93	E5 - E5
94	E6 - E6
95	E7 - E7
96	E8 - E8
97	E9 - E9
98	E10 - E10
99	E11 - E11
100	E12 - E12
101	E13 - E13
102	E14 - E14
103	E15 - E15
104	E16 - E16
105	E17 - E17
106	E18 - E18
107	E19 - E19
108	E20 - E20
109	F1 - F1
110	F2 - F2
111	F3 - F3
112	F4 - F4
113	F5 - F5
114	F6 - F6
115	F7 - F7
116	F8 - F8
117	F9 - F9
118	F10 - F10
119	F11 - F11
120	F12 - F12

Copyright 1989 Logic Modeling Systems		FILE PGA.PKG	DATE 5/25/89	PAGE # 2/2
TIME 12:50:36 pm				
LINE #	TEXT			
121	F13 - F13			
122	F14 - F14			
123	F15 - F15			
124	F16 - F16			
125	F17 - F17			
126	F18 - F18			
127	F19 - F19			
128	F20 - F20			
129	G1 - G1			
130	G2 - G2			
131	G3 - G3			
132	G4 - G4			
133	G5 - G5			
134	G6 - G6			
135	G7 - G7			
136	G8 - G8			
137	G9 - G9			
138	G10 - G10			
139	G11 - G11			
140	G12 - G12			
141	G13 - G13			
142	G14 - G14			
143	G15 - G15			
144	G16 - G16			
145	G17 - G17			
146	G18 - G18			
147	G19 - G19			
148	G20 - G20			
149	H1 - H1			
150	H2 - H2			
151	H3 - H3			
152	H4 - H4			
153	H5 - H5			
154	H6 - H6			
155	H7 - H7			
156	H8 - H8			
157	H9 - H9			
158	H10 - H10			
159	H11 - H11			
160	H12 - H12			
161	H13 - H13			
162	H14 - H14			
163	H15 - H15			
164	H16 - H16			
165	H17 - H17			
166	H18 - H18			
167	H19 - H19			
168	H20 - H20			
169	J1 - J1			
170	J2 - J2			
171	J3 - J3			
172	J4 - J4			
173	J5 - J5			
174	J6 - J6			
175	J7 - J7			
176	J8 - J8			
177	J9 - J9			
178	J10 - J10			
179	J11 - J11			
180	J12 - J12			
181	J13 - J13			
182	J14 - J14			
183	J15 - J15			
184	J16 - J16			
185	J17 - J17			
186	J18 - J18			
187	J19 - J19			
188	J20 - J20			
189	K1 - K1			
190	K2 - K2			
191	K3 - K3			
192	K4 - K4			
193	K5 - K5			
194	K6 - K6			
195	K7 - K7			
196	K8 - K8			
197	K9 - K9			
198	K10 - K10			
199	K11 - K11			
200	K12 - K12			
201	K13 - K13			
202	K14 - K14			
203	K15 - K15			
204	K16 - K16			
205	K17 - K17			
206	K18 - K18			
207	K19 - K19			
208	K20 - K20			
209	L1 - L1			
210	L2 - L2			
211	L3 - L3			
212	L4 - L4			
213	L5 - L5			
214	L6 - L6			
215	L7 - L7			
216	L8 - L8			
217	L9 - L9			
218	L10 - L10			
219	L11 - L11			
220	L12 - L12			
221	L13 - L13			
222	L14 - L14			
223	L15 - L15			
224	L16 - L16			
225	L17 - L17			
226	L18 - L18			
227	L19 - L19			
228	L20 - L20			
229	M1 - M1			
230	M2 - M2			
231	M3 - M3			
232	M4 - M4			
233	M5 - M5			
234	M6 - M6			
235	M7 - M7			
236	M8 - M8			
237	M9 - M9			
238	M10 - M10			
239	M11 - M11			
240	M12 - M12			

Copyright 1989 Logic Modeling Systems		FILE PGA.PKG	DATE 5/25/89	PAGE # 3/3
LINE #	TEXT	TIME 12:50:36 pm		
241	M13 - M13			
242	M14 - M14			
243	M15 - M15			
244	M16 - M16			
245	M17 - M17			
246	M18 - M18			
247	M19 - M19			
248	M20 - M20			
249	N1 - N1			
250	N2 - N2			
251	N3 - N3			
252	N4 - N4			
253	N5 - N5			
254	N6 - N6			
255	N7 - N7			
256	N8 - N8			
257	N9 - N9			
258	M10 - M10			
259	M11 - M11			
260	M12 - M12			
261	M13 - M13			
262	M14 - M14			
263	M15 - M15			
264	M16 - M16			
265	M17 - M17			
266	M18 - M18			
267	M19 - M19			
268	M20 - M20			
269	P1 - P1			
270	P2 - P2			
271	P3 - P3			
272	P4 - P4			
273	P5 - P5			
274	P6 - P6			
275	P7 - P7			
276	P8 - P8			
277	P9 - P9			
278	P10 - P10			
279	P11 - P11			
280	P12 - P12			
281	P13 - P13			
282	P14 - P14			
283	P15 - P15			
284	P16 - P16			
285	P17 - P17			
286	P18 - P18			
287	P19 - P19			
288	P20 - P20			
289	R1 - R1			
290	R2 - R2			
291	R3 - R3			
292	R4 - R4			
293	R5 - R5			
294	R6 - R6			
295	R7 - R7			
296	R8 - R8			
297	R9 - R9			
298	R10 - R10			
299	R11 - R11			
300	R12 - R12			
301	R13 - R13			
302	R14 - R14			
303	R15 - R15			
304	R16 - R16			
305	R17 - R17			
306	R18 - R18			
307	R19 - R19			
308	R20 - R20			
309	T1 - T1			
310	T2 - T2			
311	T3 - T3			
312	T4 - T4			
313	T5 - T5			
314	T6 - T6			
315	T7 - T7			
316	T8 - T8			
317	T9 - T9			
318	T10 - T10			
319	T11 - T11			
320	T12 - T12			
321	T13 - T13			
322	T14 - T14			
323	T15 - T15			
324	T16 - T16			
325	T17 - T17			
326	T18 - T18			
327	T19 - T19			
328	T20 - T20			
329	U1 - U1			
330	U2 - U2			
331	U3 - U3			
332	U4 - U4			
333	U5 - U5			
334	U6 - U6			
335	U7 - U7			
336	U8 - U8			
337	U9 - U9			
338	U10 - U10			
339	U11 - U11			
340	U12 - U12			
341	U13 - U13			
342	U14 - U14			
343	U15 - U15			
344	U16 - U16			
345	U17 - U17			
346	U18 - U18			
347	U19 - U19			
348	U20 - U20			
349	V1 - V1			
350	V2 - V2			
351	V3 - V3			
352	V4 - V4			
353	V5 - V5			
354	V6 - V6			
355	V7 - V7			
356	V8 - V8			
357	V9 - V9			
358	V10 - V10			
359	V11 - V11			
360	V12 - V12			

Copyright 1989
Logic Modeling Systems

FILE
PGA.PKG

DATE 5/25/89
TIME 12:50:36 pm

PAGE #
4/4

LINE #	TEXT
361	V13 = V13
362	V14 = V14
363	V15 = V15
364	V16 = V16
365	V17 = V17
366	V18 = V18
367	V19 = V19
368	V20 = V20
369	W1 = W1
370	W2 = W2
371	W3 = W3
372	W4 = W4
373	W5 = W5
374	W6 = W6
375	W7 = W7
376	W8 = W8
377	W9 = W9
378	W10 = W10
379	W11 = W11
380	W12 = W12
381	W13 = W13
382	W14 = W14
383	W15 = W15
384	W16 = W16
385	W17 = W17
386	W18 = W18
387	W19 = W19
388	W20 = W20
389	Y1 = Y1
390	Y2 = Y2
391	Y3 = Y3
392	Y4 = Y4
393	Y5 = Y5
394	Y6 = Y6
395	Y7 = Y7
396	Y8 = Y8
397	Y9 = Y9
398	Y10 = Y10
399	Y11 = Y11
400	Y12 = Y12
401	Y13 = Y13
402	Y14 = Y14
403	Y15 = Y15
404	Y16 = Y16
405	Y17 = Y17
406	Y18 = Y18
407	Y19 = Y19
408	Y20 = Y20

Copyright 1989
Logic Modeling Systems

FILE
PGA160.ADP

DATE 5/25/89
TIME 12:50:36 pm

PAGE #
1/5

LINE #	TEXT
1
2	• Copyright (c) 1988-1989 By Logic Modeling Systems, Inc.
3	• All rights reserved.
4
5	• Logic Model ADAPTER m... for 160 pin PGA adapter
6
7	[map_revision A]
8
9	adapter_mapping
10	A1 = 108 : trace_delay = 1.3663 (cut_label=1)
11	A2 = 95 : trace_delay = 1.1782 (cut_label=160)
12	A3 = 110 : trace_delay = 1.3611 (cut_label=3)
13	A4 = 92 : trace_delay = 1.3640 (cut_label=5)
14	A5 = 89 : trace_delay = 1.0955 (cut_label=8)
15	A6 = 86 : trace_delay = 1.0710 (cut_label=12)
16	A7 = 83 : trace_delay = 1.3091 (cut_label=17)
17	A8 = 80 : trace_delay = 1.0600 (cut_label=18)
18	A9 = 79 : trace_delay = 1.3474 (cut_label=20)
19	A10 = 77 : trace_delay = 1.0794 (cut_label=23)
20	A11 = 48 : trace_delay = 1.3284 (cut_label=26)
21	A12 = 72 : trace_delay = 1.1065 (cut_label=29)
22	A13 = 71 : trace_delay = 1.2980 (cut_label=34)
23	A14 = 53 : trace_delay = 1.1184 (cut_label=31)
24	A15 = 54 : trace_delay = 1.2763 (cut_label=40)
25	A16 = 55 : trace_delay = 1.0524 (cut_label=39)
26	A17 = no_connect
27	B1 = 107 : trace_delay = 1.3307 (cut_label=157)
28	B2 = 106 : trace_delay = 1.1370 (cut_label=158)
29	B3 = 109 : trace_delay = 1.1165 (cut_label=4)
30	B4 = 93 : trace_delay = 1.0910 (cut_label=6)
31	B5 = 91 : trace_delay = 1.3459 (cut_label=9)
32	B6 = 87 : trace_delay = 1.3239 (cut_label=13)
33	B7 = 84 : trace_delay = 1.0802 (cut_label=14)
34	B8 = 81 : trace_delay = 1.3783 (cut_label=15)
35	B9 = 78 : trace_delay = 1.0992 (cut_label=19)
36	B10 = 75 : trace_delay = 1.3821 (cut_label=22)
37	B11 = 49 : trace_delay = 1.1246 (cut_label=25)
38	B12 = 50 : trace_delay = 1.3916 (cut_label=28)
39	B13 = 70 : trace_delay = 1.1174 (cut_label=33)
40	B14 = 69 : trace_delay = 1.2831 (cut_label=36)
41	B15 = 66 : trace_delay = 1.0223 (cut_label=37)
42	B16 = 67 : trace_delay = 1.2789 (cut_label=38)
43	B17 = no_connect
44	C1 = 105 : trace_delay = 1.1030 (cut_label=154)
45	C2 = 104 : trace_delay = 1.3239 (cut_label=155)
46	C3 = 94 : trace_delay = 1.4476 (cut_label=159)
47	C4 = 111 : trace_delay = 1.2075 (cut_label=2)
48	C5 = 90 : trace_delay = 1.0984 (cut_label=10)
49	C6 = 88 : trace_delay = 1.4378 (cut_label=7)
50	C7 = 85 : trace_delay = 1.4175 (cut_label=11)
51	C8 = 82 : trace_delay = 1.1239 (cut_label=16)
52	C9 = 76 : trace_delay = 1.3715 (cut_label=24)
53	C10 = 74 : trace_delay = 1.1522 (cut_label=21)
54	C11 = 73 : trace_delay = 1.3891 (cut_label=30)
55	C12 = 51 : trace_delay = 1.1629 (cut_label=27)
56	C13 = 52 : trace_delay = 1.3789 (cut_label=32)
57	C14 = 68 : trace_delay = 1.1277 (cut_label=35)
58	C15 = 64 : trace_delay = 1.0609 (cut_label=43)
59	C16 = 65 : trace_delay = 1.2357 (cut_label=44)
60	C17 = 59 : trace_delay = 1.0433 (cut_label=45)
61	D1 = 103 : trace_delay = 1.1312 (cut_label=156)
62	D2 = 102 : trace_delay = 1.3293 (cut_label=151)
63	D3 = 100 : trace_delay = 1.0855 (cut_label=152)
64	D4 = 101 : trace_delay = 1.2914 (cut_label=153)
65	D5 = 55 : trace_delay = 1.2768 (cut_label=39, W1)
66	D6 = 54 : trace_delay = 1.4407 (cut_label=40)
67	D7 = 66 : trace_delay = 1.2242 (cut_label=37, W2)
68	D8 = 67 : trace_delay = 1.4445 (cut_label=38)
69	D9 = 64 : trace_delay = 1.2678 (cut_label=43, W1)
70	D10 = 57 : trace_delay = 1.2023 (cut_label=41)
71	D11 = 65 : trace_delay = 1.2465 (cut_label=44)
72	D12 = 56 : trace_delay = 1.3540 (cut_label=42)
73	D13 = 59 : trace_delay = 1.1681 (cut_label=45, W4)
74	D14 = 58 : trace_delay = 1.2728 (cut_label=46, W5)
75	D15 = 57 : trace_delay = 1.1146 (cut_label=42)
76	D16 = 56 : trace_delay = 1.2890 (cut_label=42)
77	D17 = 58 : trace_delay = 1.1655 (cut_label=46)
78	E1 = 113 : trace_delay = 1.0541 (cut_label=150)
79	E2 = 112 : trace_delay = 1.2679 (cut_label=149)
80	E3 = 98 : trace_delay = 1.2570 (cut_label=147)
81	E4 = 148 : trace_delay = 1.1905 (cut_label=121, W18)
82	E5 = 53 : trace_delay = 1.2736 (cut_label=31)
83	E6 = 69 : trace_delay = 1.4195 (cut_label=36)
84	E7 = 68 : trace_delay = 1.2448 (cut_label=35)
85	E8 = 58 : trace_delay = 1.3481 (cut_label=46, W5)
86	E9 = 46 : trace_delay = 1.0855 (cut_label=51)
87	E10 = 62 : trace_delay = 1.1798 (cut_label=47, W6)
88	E11 = 61 : trace_delay = 1.3227 (cut_label=48)
89	E12 = 63 : trace_delay = 1.0351 (cut_label=53)
90	E13 = 60 : trace_delay = 1.0823 (cut_label=49)
91	E14 = 62 : trace_delay = 1.1171 (cut_label=47, W6)
92	E15 = 61 : trace_delay = 1.2543 (cut_label=48)
93	E16 = 60 : trace_delay = 1.0229 (cut_label=49)
94	E17 = 62 : trace_delay = 1.0157 (cut_label=47)
95	F1 = 99 : trace_delay = 1.0395 (cut_label=146)
96	F2 = 97 : trace_delay = 1.0946 (cut_label=142)
97	F3 = 114 : trace_delay = 1.0745 (cut_label=148)
98	F4 = 158 : trace_delay = 1.3379 (cut_label=122)
99	F5 = 159 : trace_delay = 1.4392 (cut_label=123)
100	F6 = 45 : trace_delay = 1.4630 (cut_label=50)
101	F7 = 4 : trace_delay = 1.3005 (cut_label=70)
102	F8 = 11 : trace_delay = 1.3623 (cut_label=64)
103	F9 = 40 : trace_delay = 1.3152 (cut_label=58)
104	F10 = 43 : trace_delay = 1.3254 (cut_label=54)
105	F11 = 44 : trace_delay = 1.0445 (cut_label=55)
106	F12 = 47 : trace_delay = 1.2466 (cut_label=52)
107	F13 = 45 : trace_delay = 1.3248 (cut_label=50)
108	F14 = 46 : trace_delay = 1.0037 (cut_label=51)
109	F15 = 47 : trace_delay = 1.1836 (cut_label=52)
110	F16 = 63 : trace_delay = 0.9594 (cut_label=53)
111	F17 = 46 : trace_delay = 0.9541 (cut_label=51)
112	G1 = 115 : trace_delay = 1.0182 (cut_label=144)
113	G2 = 96 : trace_delay = 1.2366 (cut_label=143)
114	G3 = 117 : trace_delay = 1.2401 (cut_label=141)
115	G4 = 147 : trace_delay = 1.1373 (cut_label=119, W19)
116	G5 = 146 : trace_delay = 1.3441 (cut_label=120)
117	G6 = 124 : trace_delay = 1.3129 (cut_label=102)
118	G7 = 27 : trace_delay = 1.1173 (cut_label=78)
119	G8 = 0 : trace_delay = 1.2701 (cut_label=74)
120	G9 = 10 : trace_delay = 1.2299 (cut_label=65)

Copyright 1989
Logic Modeling Systems

FILE
PGA160.ADP

DATE 5/25/89
TIME 12:50:36 pm

PAGE #
2/6

LINE #		TEXT
121	G10 = 31	: trace_delay = 1.2314 (cut_label=61)
122	G11 = 30	: trace_delay = 1.3637 (cut_label=60)
123	G12 = 41	: trace_delay = 1.2358 (cut_label=57, W7)
124	G13 = 42	: trace_delay = 1.2017 (cut_label=56)
125	G14 = 43	: trace_delay = 1.2365 (cut_label=54)
126	G15 = 44	: trace_delay = 0.9797 (cut_label=55)
127	G16 = 45	: trace_delay = 1.2768 (cut_label=50)
128	G17 = 43	: trace_delay = 1.1770 (cut_label=54)
129	H1 = 116	: trace_delay = 1.2099 (cut_label=145)
130	H2 = 119	: trace_delay = 1.2284 (cut_label=139)
131	H3 = 118	: trace_delay = 1.0844 (cut_label=140)
132	H4 = 135	: trace_delay = 1.2625 (cut_label=138)
133	H5 = 134	: trace_delay = 1.0432 (cut_label=117)
134	H6 = 122	: trace_delay = 1.1301 (cut_label=101)
135	H7 = 19	: trace_delay = 1.2035 (cut_label=88)
136	H8 = 23	: trace_delay = 1.1200 (cut_label=82)
137	H9 = 5	: trace_delay = 1.4600 (cut_label=71)
138	H10 = 8	: trace_delay = 1.4349 (cut_label=67)
139	H11 = 12	: trace_delay = 1.3966 (cut_label=62)
140	H12 = 13	: trace_delay = 1.1104 (cut_label=63)
141	H13 = 29	: trace_delay = 1.1661 (cut_label=59)
142	H14 = 40	: trace_delay = 1.2310 (cut_label=58)
143	H15 = 41	: trace_delay = 1.0217 (cut_label=57)
144	H16 = 42	: trace_delay = 1.1508 (cut_label=56)
145	H17 = 40	: trace_delay = 1.1680 (cut_label=58)
146	J1 = 149	: trace_delay = 1.0438 (cut_label=138)
147	J2 = 151	: trace_delay = 1.2065 (cut_label=137)
148	J3 = 150	: trace_delay = 1.0527 (cut_label=136)
149	J4 = 137	: trace_delay = 1.4432 (cut_label=116, W17)
150	J5 = 145	: trace_delay = 1.0707 (cut_label=113, W15)
151	J6 = 143	: trace_delay = 1.2896 (cut_label=104)
152	J7 = 37	: trace_delay = 1.4443 (cut_label=97)
153	J8 = 18	: trace_delay = 1.3952 (cut_label=91)
154	J9 = 22	: trace_delay = 1.4829 (cut_label=85, W11)
155	J10 = 6	: trace_delay = 1.3476 (cut_label=72, W9)
156	J11 = 6	: trace_delay = 1.2071 (cut_label=68)
157	J12 = 9	: trace_delay = 1.2998 (cut_label=64, W8)
158	J13 = 11	: trace_delay = 1.2718 (cut_label=64)
159	J14 = 31	: trace_delay = 1.1453 (cut_label=61)
160	J15 = 30	: trace_delay = 1.2747 (cut_label=60)
161	J16 = 29	: trace_delay = 1.0982 (cut_label=59)
162	J17 = 31	: trace_delay = 1.0859 (cut_label=61)
163	K1 = 132	: trace_delay = 1.2620 (cut_label=135)
164	K2 = 133	: trace_delay = 1.0777 (cut_label=134)
165	K3 = 153	: trace_delay = 1.2369 (cut_label=133)
166	K4 = 139	: trace_delay = 1.2711 (cut_label=112)
167	K5 = 142	: trace_delay = 1.3450 (cut_label=110, W14)
168	K6 = 125	: trace_delay = 1.0725 (cut_label=107)
169	K7 = 120	: trace_delay = 1.1257 (cut_label=99)
170	K8 = 16	: trace_delay = 1.1620 (cut_label=92)
171	K9 = 32	: trace_delay = 1.1621 (cut_label=86)
172	K10 = 26	: trace_delay = 1.2063 (cut_label=81, W10)
173	K11 = 15	: trace_delay = 1.1050 (cut_label=76)
174	K12 = 7	: trace_delay = 1.1860 (cut_label=73)
175	K13 = 10	: trace_delay = 1.1616 (cut_label=69)
176	K14 = 10	: trace_delay = 1.1052 (cut_label=65)
177	K15 = 12	: trace_delay = 1.1015 (cut_label=62)
178	K16 = 13	: trace_delay = 1.0398 (cut_label=63)
179	K17 = 10	: trace_delay = 1.0422 (cut_label=65)
180	L1 = 152	: trace_delay = 1.0898 (cut_label=132)
181	L2 = 130	: trace_delay = 1.3498 (cut_label=131)
182	L3 = 131	: trace_delay = 1.0785 (cut_label=130)
183	L4 = 136	: trace_delay = 1.0632 (cut_label=115)
184	L5 = 141	: trace_delay = 1.0475 (cut_label=109)
185	L6 = 121	: trace_delay = 1.0793 (cut_label=103)
186	L7 = 38	: trace_delay = 1.1975 (cut_label=96)
187	L8 = 35	: trace_delay = 1.4054 (cut_label=95)
188	L9 = 33	: trace_delay = 1.3370 (cut_label=89)
189	L10 = 25	: trace_delay = 1.0873 (cut_label=80)
190	L11 = 28	: trace_delay = 1.2647 (cut_label=79)
191	L12 = 1	: trace_delay = 1.3942 (cut_label=78)
192	L13 = 4	: trace_delay = 1.1827 (cut_label=70)
193	L14 = 8	: trace_delay = 1.3547 (cut_label=67)
194	L15 = 9	: trace_delay = 1.1110 (cut_label=64)
195	L16 = 11	: trace_delay = 1.2179 (cut_label=64)
196	L17 = 8	: trace_delay = 1.2953 (cut_label=67)
197	M1 = 128	: trace_delay = 1.1914 (cut_label=129)
198	M2 = 129	: trace_delay = 1.0395 (cut_label=128)
199	M3 = 156	: trace_delay = 1.2802 (cut_label=127)
200	M4 = 138	: trace_delay = 0.9886 (cut_label=111)
201	M5 = 127	: trace_delay = 1.0316 (cut_label=105)
202	M6 = 126	: trace_delay = 1.2711 (cut_label=104)
203	M7 = 121	: trace_delay = 1.3101 (cut_label=100, W13)
204	M8 = 36	: trace_delay = 1.0730 (cut_label=94)
205	M9 = 34	: trace_delay = 1.0914 (cut_label=96)
206	M10 = 20	: trace_delay = 1.2434 (cut_label=84, W12)
207	M11 = 24	: trace_delay = 1.3075 (cut_label=83)
208	M12 = 14	: trace_delay = 1.2795 (cut_label=77)
209	M13 = 0	: trace_delay = 1.1663 (cut_label=74)
210	M14 = 5	: trace_delay = 1.3664 (cut_label=71)
211	M15 = 6	: trace_delay = 1.1270 (cut_label=68)
212	M16 = 7	: trace_delay = 1.2877 (cut_label=69)
213	M17 = 5	: trace_delay = 1.3168 (cut_label=71)
214	N1 = 157	: trace_delay = 1.0902 (cut_label=126)
215	N2 = 155	: trace_delay = 1.3015 (cut_label=125)
216	N3 = 154	: trace_delay = 1.1742 (cut_label=124)
217	N4 = 144	: trace_delay = 1.2432 (cut_label=114, W16)
218	N5 = 140	: trace_delay = 1.1839 (cut_label=108)
219	N6 = 124	: trace_delay = 1.2124 (cut_label=102)
220	N7 = 122	: trace_delay = 1.0481 (cut_label=101)
221	N8 = 39	: trace_delay = 1.3787 (cut_label=96)
222	N9 = 17	: trace_delay = 1.3669 (cut_label=93)
223	N10 = 19	: trace_delay = 1.0861 (cut_label=88)
224	N11 = 21	: trace_delay = 1.2765 (cut_label=87)
225	N12 = 27	: trace_delay = 1.0035 (cut_label=78)
226	N13 = 23	: trace_delay = 1.0163 (cut_label=82)
227	N14 = 2	: trace_delay = 1.2413 (cut_label=72, W9)
228	N15 = 3	: trace_delay = 1.3049 (cut_label=73)
229	N16 = 4	: trace_delay = 1.1288 (cut_label=70)
230	N17 = 2	: trace_delay = 1.1403 (cut_label=72)
231	P1 = 159	: trace_delay = 1.2588 (cut_label=123)
232	P2 = 146	: trace_delay = 1.1932 (cut_label=120)
233	P3 = 134	: trace_delay = 0.9133 (cut_label=117)
234	P4 = 145	: trace_delay = 0.9906 (cut_label=113, W15)
235	P5 = 142	: trace_delay = 1.2623 (cut_label=110, W14)
236	P6 = 125	: trace_delay = 1.0088 (cut_label=107)
237	P7 = 143	: trace_delay = 1.1822 (cut_label=104)
238	P8 = 120	: trace_delay = 1.0426 (cut_label=99)
239	P9 = 37	: trace_delay = 1.3590 (cut_label=97)
240	P10 = 16	: trace_delay = 1.0896 (cut_label=92)

Copyright 1989 Logic Modeling Systems		FILE PGA160.ADP	DATE 5/25/89	PAGE # 3/7
			TIME 12:50:36 pm	
LINE #	TEXT			
241	P11 = 18	: trace_delay = 1.3074	{, cut_label=91 }	
242	P12 = 32	: trace_delay = 1.0884	{, cut_label=86 }	
243	P13 = 22	: trace_delay = 1.3817	{, cut_label=85, W11 }	
244	P14 = 26	: trace_delay = 1.3126	{, cut_label=81, W10 }	
245	P15 = 15	: trace_delay = 0.9987	{, cut_label=76 }	
246	P16 = 1	: trace_delay = 1.3203	{, cut_label=75 }	
247	P17 = 26	: trace_delay = 1.4148	{, cut_label=81 }	
248	R1 = 158	: trace_delay = 1.1408	{, cut_label=112 }	
249	R2 = 147	: trace_delay = 0.9557	{, cut_label=119 }	
250	R3 = 137	: trace_delay = 1.1913	{, cut_label=116 }	
251	R4 = 139	: trace_delay = 1.1872	{, cut_label=112 }	
252	R5 = 141	: trace_delay = 0.9822	{, cut_label=109 }	
253	R6 = 126	: trace_delay = 1.2074	{, cut_label=106 }	
254	R7 = 123	: trace_delay = 0.9962	{, cut_label=103 }	
255	R8 = 121	: trace_delay = 1.5086	{, cut_label=100 }	
256	R9 = 38	: trace_delay = 1.1221	{, cut_label=96 }	
257	R10 = 35	: trace_delay = 1.3193	{, cut_label=95 }	
258	R11 = 34	: trace_delay = 1.0334	{, cut_label=90 }	
259	R12 = 33	: trace_delay = 1.2521	{, cut_label=89 }	
260	R13 = 20	: trace_delay = 1.0062	{, cut_label=84 }	
261	R14 = 25	: trace_delay = 0.9809	{, cut_label=80 }	
262	R15 = 28	: trace_delay = 1.1805	{, cut_label=79 }	
263	R16 = 0	: trace_delay = 1.1024	{, cut_label=74 }	
264	R17 = no_connect			
265	T1 = 148	: trace_delay = 0.9435	{, cut_label=121 }	
266	T2 = 135	: trace_delay = 1.1249	{, cut_label=118 }	
267	T3 = 136	: trace_delay = 0.9555	{, cut_label=115 }	
268	T4 = 138	: trace_delay = 0.9238	{, cut_label=111 }	
269	T5 = 140	: trace_delay = 1.1371	{, cut_label=108 }	
270	T6 = 127	: trace_delay = 0.9486	{, cut_label=105 }	
271	T7 = 124	: trace_delay = 1.1644	{, cut_label=102 }	
272	T8 = 122	: trace_delay = 0.9967	{, cut_label=101 }	
273	T9 = 39	: trace_delay = 1.3178	{, cut_label=98 }	
274	T10 = 36	: trace_delay = 1.0803	{, cut_label=94 }	
275	T11 = 17	: trace_delay = 1.3130	{, cut_label=93 }	
276	T12 = 19	: trace_delay = 1.0177	{, cut_label=88 }	
277	T13 = 21	: trace_delay = 1.2226	{, cut_label=87 }	
278	T14 = 24	: trace_delay = 1.2249	{, cut_label=83 }	
279	T15 = 27	: trace_delay = 0.9396	{, cut_label=78 }	
280	T16 = 14	: trace_delay = 1.1958	{, cut_label=77 }	
281	T17 = no_connect			
282	U1 = no_connect			
283	U2 = no_connect			
284	U3 = 144	: trace_delay = 1.1174	{, cut_label=114 }	
285	U4 = 145	: trace_delay = 0.8862	{, cut_label=113 }	
286	U5 = 142	: trace_delay = 1.1597	{, cut_label=110 }	
287	U6 = 125	: trace_delay = 0.9620	{, cut_label=107 }	
288	U7 = 143	: trace_delay = 1.1228	{, cut_label=104 }	
289	U8 = 120	: trace_delay = 0.9832	{, cut_label=99 }	
290	U9 = 37	: trace_delay = 1.2996	{, cut_label=97 }	
291	U10 = 16	: trace_delay = 1.0246	{, cut_label=92 }	
292	U11 = 18	: trace_delay = 1.2606	{, cut_label=91 }	
293	U12 = 32	: trace_delay = 0.9972	{, cut_label=86 }	
294	U13 = 22	: trace_delay = 1.2260	{, cut_label=85 }	
295	U14 = 23	: trace_delay = 0.9316	{, cut_label=82 }	
296	U15 = no_connect			
297	U16 = no_connect			
298	U17 = no_connect			

APPENDIX II

Table of Contents

AM7990	Tab 1
DIP	Tab 2

Copyright 1989 Logic Modeling Systems		FILE AM7990.MDL	DATE 5/25/89	PAGE # 1/2
			TIME 12:46:58 pm	
LINE #	TEXT			
1	[.....]			
2	[* Copyright (c) 1988-1989 by Logic Modeling Systems, Inc.			
3	[* All rights reserved.			
4	[.....]			
5	[* Model main MODEL file for Advanced Micro Devices AM7990			
6	[* Local Area Network Controller For Ethernet			
7	[.....]			
8	device_name AM7990			
9	[.....]			
10	[model_revision A]			
11	[shell_revision A]			
12	[.....]			
13	physical 'AM7990.DEV'			
14	timing 'AM7990.TWG'			
15	package 'DIP48.PKG' [48]			
16	adapter 'DIP76.ADP'			
17	options 'AM7990.OPT'			
18	names 'AM7990.NAM'			
19	[.....]			

Copyright 1989

Logic Modeling Systems

FILE

AM7990.DEV

DATE

5/25/89

PAGE #

TIME

12:46:58 pm

1/1

```

LINE #      TEXT
1  { .....
2  * Copyright (c) 1988 by Logic Modeling Systems, Inc. All rights reserved. *
3  * .....
4  * Logic Model DEVICE file for Advanced Micro Devices AM7990
5  * Local Area Network Controller For Ethernet
6  * .....
7  * Data obtained from October 1988 Data Sheet 05498B/0
8  * .....
9  { model_revision A }
10 { shell_revision A }
11 { .....
12 technology      NMOS
13 { .....
14 device_speed     99 nS - 101 nS      { AM7990PC, AM7990DC }
15 device_setup_time 40                  { parameter 18 }
16 device_hold_time  5                   { parameter 17 }
17 { .....
18 in_pin
19   RXCLK           = 27 : store          { receive clock }
20   TXCLK           = 25 : store          { transmit clock }
21   RX              = 31                  { receive data }
22   '-CS'           = 20                  { chip select }
23   ADDR            = 21                  { register address }
24   '-RESET'        = 23                  { reset }
25   CLSN            = 28                  { collision }
26   RENA            = 30                  { receiver enable }
27   '-HLD'          = 19 : store          { bus hold acknowledge }
28 { .....
29 out_pin
30   ALE             = 18                  { address latch enable }
31   '-INTR'         = 11                  { interrupt }
32   TX              = 29                  { transmit data }
33   TEMA            = 26                  { transmitter enable }
34   '-BML/BUSAKO'   = 16                  { byte mask }
35   '-BMO/BYTE'     = 15                  { byte mask }
36   '-DALI'         = 12                  { enable data in }
37   '-DALO'         = 13                  { enable data out }
38   A(23:16)        = 32,33,34,35,36,37,38,39 { high address byte }
39 { .....
40 io_pin
41   DAL(15:0)       = 40,41,42,43,44,45,46,47, { data/address bus }
42   2, 3, 4, 5, 6, 7, 8, 9
43   READ            = 10                  { read transfer }
44   '-DAS'          = 14 : store          { data strobe }
45   '-READY'        = 22 : feedback      { acknowledge }
46   '-BOLD/BUSRQ'   = 17                  { bus hold/request }
47 { .....
48 power_pin
49   VCC             = 48
50 { .....
51 ground_pin
52   VSS             = 1, 24
53 { .....
54 initialize
55 { operation : reset wait, ->CSRL wait CSRL=0000 wait }
56   CLSN, RENA = 3(0,0), 3(0,0), 8(0,0), 0,0,0,0, fall(0,0), 0,0,0,0
57   TXCLK, RXCLK = 3(0,1), 3(0,1), 8(0,1), 0,1,0,1, fall(0,1), 0,1,0,1
58   '-RESET' = 3(0,0), 3(1,1), 8(1,1), 1,1,1,1, fall(1,1), 1,1,1,1
59   DAL(15:1) = 3(0,0), 3(0,0), 8(0,0), 0,0,0,0, fall(0,0), 0,0,0,0
60   DAL(0) = 3(1,1), 3(1,1), 8(1,1), 1,1,0,0, fall(0,0), 0,0,0,0
61   '-DAS' = 3(1,1), 3(1,1), 8(0,0), 1,1,1,1, fall(0,0), 1,1,1,1
62   ADDR = 3(0,0), 3(0,0), 8(1,1), 1,1,0,0, fall(0,0), 0,0,0,0
63   READ = 3(0,0), 3(0,0), 8(0,0), 0,0,0,0, fall(0,0), 0,0,0,0
64   '-CS' = 3(1,1), 3(0,0), 8(0,0), 0,0,0,0, fall(0,0), 0,0,1,1
65   '-READY' = 3(1,1), 3(1,1), 8(1,1), 1,1,1,1, fall(1,1), 1,1,1,1
66 { .....

```


Copyright 1989
Logic Modeling Systems

FILE
AM7990.TMG

DATE 5/25/89
TIME 12:46:58 pm

PAGE #
1/5

LINE #	TEXT
1	{ }
2	{* Copyright (c) 1988 by Logic Modeling Systems, Inc. All rights reserved. *}
3	{ }
4	{* Logic Model TIMING file for Advanced Micro Devices AM7990}
5	{* Local Area Network Controller for Ethernet}
6	{ }
7	{* Data from AMD October 1986 datasheet 05698 rev B}
8	{ }
9	{ model_revision A }
10	{ shell_revision A }
11	{ }
12	default_delay min 0, max 35
13	delay
14	from valid('HOLD')
15	to valid('HOLD/BUSREQ') = MAX 75 (UNSPEC)
16	
17	from high(TCLK)
18	to valid(TENA) = 5, 70 (6, 7)
19	to valid(TX) = 5, 70 (8, 9)
20	to any('HOLD/BUSREQ') = MAX 95 (65)
21	to valid(A[23:16]) = MAX 100 (66, 67)
22	to float(A[23:16]) = MAX 95 (67)
23	to valid('BML/BUSAKO') = MAX 75 (21, 65)
24	to float('BML/BUSAKO') = MAX 95 (67)
25	to valid('BMO/BYTE') = MAX 95 (21, 65)
26	to float('BMO/BYTE') = MAX 95 (67)
27	to valid(DAL[15:0]) = MAX 75 (21, 65)
28	to float(DAL[15:0]) = MAX 95 (67)
29	to valid('DALO') = MAX 75 (21, 65)
30	to float('DALO') = MAX 95 (67)
31	to valid(READ) = MAX 75 (21, 65)
32	to float(READ) = MAX 95 (67)
33	to valid(ALE) = MAX 75 (21, 65)
34	to float(ALE) = MAX 95 (67)
35	to valid('DAS') = MAX 75 (71)
36	to high('DAS') = MAX 90 (21, 65)
37	to float('DAS') = MAX 95 (67)
38	to valid('DALI') = MAX 75 (21, 65)
39	to float('DALI') = MAX 95 (67)
40	to low('READY') = MAX 75 (21, 65)
41	from low(TCLK)
42	to low(ALE) = MAX 90 (68)
43	to low('DAS') = MAX 90 (69)
44	to valid(DAL[15:0]) = MAX 75 (UNSPEC)
45	to valid(A[23:16]) = MAX 75 (UNSPEC)
46	to valid(READ) = MAX 75 (UNSPEC)
47	to valid('BML/BUSAKO') = MAX 75 (UNSPEC)
48	to valid('BMO/BYTE') = MAX 75 (UNSPEC)
49	to valid('DALI') = MAX 75 (UNSPEC)
50	to valid('DALO') = MAX 75 (UNSPEC)
51	from valid(TCLK)
52	to any('INTR') = MAX 75 (UNSPEC)
53	
54	from high('DAS')
55	to float(DAL[15:0]) = 0, 35 (37)
56	to float('READY') = 0, 35 (62)
57	

Copyright 1989
Logic Modeling Systems

FILE
DIP48.PKG

DATE 5/25/89
TIME 12:48:16 pm

PAGE #
1/1

LINE #	TEXT
1
2	Copyright (c) 1988 by Logic Modeling Systems, Inc. All rights reserved. *
3
4	* Logic Model PACKAGE map file for 48 pin dual inline * 755
5
6	{ map_revision A }
7
8	package_mapping
9
10	1 = 1
11	2 = 2
12	3 = 3
13	4 = 4
14	5 = 5
15	6 = 6
16	7 = 7
17	8 = 8
18	9 = 9
19	10 = 10
20	11 = 11
21	12 = 12
22	13 = 13
23	14 = 14
24	15 = 15
25	16 = 16
26	17 = 17
27	18 = 18
28	19 = 19
29	20 = 20
30	21 = 21
31	22 = 22
32	23 = 23
33	24 = 24
34	25 = 25
35	26 = 26
36	27 = 27
37	28 = 28
38	29 = 29
39	30 = 30
40	31 = 31
41	32 = 32
42	33 = 33
43	34 = 34
44	35 = 35
45	36 = 36
46	37 = 37
47	38 = 38
48	39 = 39
49	40 = 40
50	41 = 41
51	42 = 42
52	43 = 43
53	44 = 44
54	45 = 45
55	46 = 46
56	47 = 47
57	48 = 48

Copyright 1989 Logic Modeling Systems		FILE DIP76.ADP	DATE 5/25/89	PAGE # 1/2
TIME 12:48:16 pm				
LINE #	TEXT			
1	*****			
2	* Copyright (c) 1988-1989 by Logic Modeling Systems, Inc.			
3	* All rights reserved.			
4	*****			
5	* Logic Model ADAPTER map file for 76 pin DIP adapter			
6	*****			
7	map_revision A)			
8	*****			
9	adapter_mapping			
10	1 = 74	: trace_delay = 0.5670	(cut_label=1)
11	2 = 75	: trace_delay = 0.5523	(cut_label=2)
12	3 = 49	: trace_delay = 0.5553	(cut_label=3)
13	4 = 51	: trace_delay = 0.5740	(cut_label=4)
14	5 = 50	: trace_delay = 0.5602	(cut_label=5)
15	6 = 53	: trace_delay = 0.5789	(cut_label=6)
16	7 = 52	: trace_delay = 0.5665	(cut_label=7)
17	8 = 68	: trace_delay = 0.5740	(cut_label=8)
18	9 = 66	: trace_delay = 0.5988	(cut_label=9)
19	10 = 67	: trace_delay = 0.5866	(cut_label=10)
20	11 = 57	: trace_delay = 0.6014	(cut_label=11)
21	12 = 56	: trace_delay = 0.5823	(cut_label=12)
22	13 = 59	: trace_delay = 0.5912	(cut_label=13)
23	14 = 62	: trace_delay = 0.6131	(cut_label=14)
24	15 = 61	: trace_delay = 0.6040	(cut_label=15)
25	16 = 46	: trace_delay = 0.6337	(cut_label=16)
26	17 = 42	: trace_delay = 0.6822	(cut_label=17)
27	18 = 41	: trace_delay = 0.6879	(cut_label=18)
28	19 = 39	: trace_delay = 0.7084	(cut_label=19)
29	20 = 17	: trace_delay = 0.6411	(cut_label=20)
30	21 = 19	: trace_delay = 0.6204	(cut_label=21)
31	22 = 20	: trace_delay = 0.5925	(cut_label=22)
32	23 = 22	: trace_delay = 0.6184	(cut_label=23)
33	24 = 24	: trace_delay = 0.5949	(cut_label=24)
34	25 = 25	: trace_delay = 0.6050	(cut_label=25)
35	26 = 27	: trace_delay = 0.5819	(cut_label=26)
36	27 = 15	: trace_delay = 0.5744	(cut_label=27)
37	28 = 14	: trace_delay = 0.5868	(cut_label=28)
38	29 = 30	: trace_delay = 0.5644	(cut_label=29)
39	30 = 31	: trace_delay = 0.5777	(cut_label=30)
40	31 = 12	: trace_delay = 0.5584	(cut_label=31)
41	32 = 9	: trace_delay = 0.5808	(cut_label=32)
42	33 = 8	: trace_delay = 0.5925	(cut_label=33)
43	34 = 5	: trace_delay = 0.5835	(cut_label=34)
44	35 = 4	: trace_delay = 0.5694	(cut_label=35)
45	36 = 2	: trace_delay = 0.5513	(cut_label=36)
46	37 = 1	: trace_delay = 0.5656	(cut_label=37)
47	38 = 0	: trace_delay = 0.5530	(cut_label=38)
48	39 = 3	: trace_delay = 0.6306	(cut_label=39)
49	40 = 6	: trace_delay = 0.6413	(cut_label=40)
50	41 = 7	: trace_delay = 0.6186	(cut_label=41)
51	42 = 11	: trace_delay = 0.6414	(cut_label=42)
52	43 = 10	: trace_delay = 0.6249	(cut_label=43)
53	44 = 13	: trace_delay = 0.6444	(cut_label=44)
54	45 = 29	: trace_delay = 0.6623	(cut_label=45)
55	46 = 28	: trace_delay = 0.6640	(cut_label=46)
56	47 = 26	: trace_delay = 0.6607	(cut_label=47)
57	48 = 23	: trace_delay = 0.6838	(cut_label=48)
58	49 = 21	: trace_delay = 0.6947	(cut_label=49)
59	50 = 32	: trace_delay = 0.7195	(cut_label=50)
60	51 = 33	: trace_delay = 0.6913	(cut_label=51)
61	52 = 18	: trace_delay = 0.7271	(cut_label=52)
62	53 = 34	: trace_delay = 0.7125	(cut_label=53)
63	54 = 16	: trace_delay = 0.7343	(cut_label=54)
64	55 = 35	: trace_delay = 0.7223	(cut_label=55)
65	56 = 36	: trace_delay = 0.7018	(cut_label=56)
66	57 = 37	: trace_delay = 0.7238	(cut_label=57)
67	58 = 44	: trace_delay = 0.6953	(cut_label=58)
68	59 = 63	: trace_delay = 0.6960	(cut_label=59)
69	60 = 47	: trace_delay = 0.6977	(cut_label=60)
70	61 = 60	: trace_delay = 0.6919	(cut_label=61)
71	62 = 58	: trace_delay = 0.6705	(cut_label=62)
72	63 = 65	: trace_delay = 0.6623	(cut_label=63)
73	64 = 64	: trace_delay = 0.6689	(cut_label=64)
74	65 = 54	: trace_delay = 0.6465	(cut_label=65)
75	66 = 55	: trace_delay = 0.6589	(cut_label=66)
76	67 = 69	: trace_delay = 0.6375	(cut_label=67)
77	68 = 71	: trace_delay = 0.6249	(cut_label=68)
78	69 = 70	: trace_delay = 0.6373	(cut_label=69)
79	70 = 73	: trace_delay = 0.6159	(cut_label=70)
80	71 = 72	: trace_delay = 0.6339	(cut_label=71)
81	72 = 48	: trace_delay = 0.6066	(cut_label=72)
82	73 = 76	: trace_delay = 0.5984	(cut_label=73)
83	74 = 77	: trace_delay = 0.6114	(cut_label=74)
84	75 = 79	: trace_delay = 0.5897	(cut_label=75)
85	76 = 78	: trace_delay = 0.6015	(cut_label=76)
86	77 = 45 : extra			
87	78 = 43 : extra			
88	79 = 40 : extra			
89	80 = 38 : extra			

APPENDIX III

Table of Contents

lm	Tab 1
include	Tab 2
sfi	Tab 3
networkh	Tab 4
networkm	Tab 5
networkc	Tab 6
shellswh	Tab 7
shellswm	Tab 8
shellswc	Tab 9
diags	Tab 10
bsp.diags	Tab 11
tasks.diags	Tab 12
lml000	Tab 13
bsp.lml000	Tab 14
tasks.lml000	Tab 15
os	Tab 16
misc	Tab 17

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
1/1

```

LINE #          SOURCE TEXT
1  #include <stdio.h>
2  #include <ctype.h>
3  #include "common.h"
4  #include "lm_sfl.h"
5  #include "lm_text.h"
6  #include "common.h"
7  #include "network.h"
8
9  #define MIN_PORT 1024
10 #define MAX_PORT 0x10000
11 #define MIN_SEG 0
12 #define MAX_SEG 32
13 #define MIN_SLOT 0
14 #define MAX_SLOT 7
15 #define MIN_LANE 'a'
16 #define MAX_LANE 'd'
17
18 extern char error_prefix[];
19
20
21 user_number_check (user_number_str)
22     char *user_number_str;
23 {
24     long number_of_users;
25     long *list_of_users;
26     long user_number, loop;
27     char **list_of_user_names, **list_of_host_names;
28
29     if (FAILURE == integer_check (user_number_str))
30         return (FAILURE);
31
32     user_number = atol (user_number_str);
33
34     if ((user_number < 0) || (user_number > MAX_USERS)) {
35         (void) fprintf (stderr, "%s: invalid user number %d\n", error_prefix, user_number);
36         return (FAILURE);
37     }
38
39     LM_CHECK (lm_inquire_user_list (&number_of_users, &list_of_users,
40                                     &list_of_host_names, &list_of_user_names));
41
42
43     for (loop = 0; loop < number_of_users; loop++, list_of_users++) {
44         if (*list_of_users == user_number)
45             return (SUCCESS);
46     }
47
48     (void) fprintf (stderr, "%s: there is no user number %d\n", error_prefix, user_number);
49     return (FAILURE);
50 }
51
52
53 check_users (modeler_name)
54     char *modeler_name;
55 {
56     /* check to see if there are other users on the system */
57
58     long number_of_users, others;
59     long *list_of_users;
60     long status;
61     char **list_of_user_names, **list_of_host_names;
62     char reply[80];
63
64     status = lm_select_modeler(modeler_name);
65
66     if (LM_SUCCESS != status) {
67         /* I know this seems weird, but if we can't
68          * select the modeler, assume that there is
69          * no one on the modeler.
70          */
71         return (SUCCESS);
72     }
73
74     /* get the total number of users on the system */
75
76     LM_CHECK (lm_inquire_user_list (&number_of_users, &list_of_users,
77                                     &list_of_host_names, &list_of_user_names));
78
79     /* if there are other users on the system, give the operator the
80      * option of quitting out.
81      */
82     if ((others = number_of_users - 1) > 0) {
83         (void) printf((others == 1)? "There is %d other user on the system. \n":
84                      "There are %d other users on the system. \n", others);
85
86         lm_get_input ("Do you wish to continue? (y/n) ", reply, sizeof(reply));
87         while (FAILURE == yes_or_no (reply)) {
88             lm_get_input ("Do you wish to continue? (y/n) ",
89                          reply, sizeof(reply));
90         }
91
92         if ((reply[0] == 'Y') || (reply[0] == 'y')) return (SUCCESS);
93         return (FAILURE);
94     }
95
96     (void) printf("There are no active users on modeler \"%s\" \n", modeler_name);
97     return (SUCCESS);
98 }
99
100
101 is_ident(c)
102     char c;
103 {
104
105     if (
106         ((c >= 'A') && (c <= 'Z')) ||
107         ((c >= 'a') && (c <= 'z')) ||
108         ((c >= '0') && (c <= '9')) ||
109         (c == '.') || (c == '@') ||
110         (c == '$') || (c == '%') ||
111         (c == '+') || (c == '-') ||
112         (c == '/') || (c == '_') ||
113         (c == '(') || (c == ')') ) return (TRUE);
114
115     return (FALSE);
116 }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
2/2

LINE # SOURCE TEXT

```

121 )
122
123
124 is_space(c)
125     char c;
126 {
127     if (isspace(c)) return (SUCCESS);
128     return (FAILURE);
129 }
130
131 is_digit(c)
132     char c;
133 {
134     if ((c >= '0') && (c <= '9')) return (SUCCESS);
135     return (FAILURE);
136 }
137
138 host_check (host_name)
139     char *host_name;
140 {
141     u_long internet_address;
142
143     if (FAILURE == get_internet_address(host_name, &internet_address)) {
144         (void) fprintf(stderr, "%s: unknown host: %s \n",
145             error_prefix, host_name);
146         return(FAILURE);
147     }
148     return(SUCCESS);
149 }
150
151 integer_check(string)
152     char *string;
153 {
154     int loop;
155
156     if (strlen(string) == 0) {
157         return(FAILURE);
158     }
159     for (loop = 0; loop < strlen(string); loop++) {
160         if (FAILURE == is_digit(string[loop])) {
161             (void) fprintf(stderr,
162                 "%s: value must be numeric: %s \n",
163                 error_prefix, string);
164             return(FAILURE);
165         }
166     }
167
168     if ((strlen(string)>10) ||
169         ((strlen(string) == 10) &&
170          (str_cmp(string, "2147483647") > 0))) {
171         (void) fprintf(stderr,
172             "%s: integer value too large: %s \n",
173             error_prefix, string);
174         return(FAILURE);
175     }
176     return(SUCCESS);
177 }
178
179 true_or_false(string)
180     char *string;
181 {
182     if (strlen(string) == 0) {
183         return(FAILURE);
184     }
185     if ((0 != strncmp ("true", string, strlen(string))) &&
186         (0 != strncmp("false", string, strlen(string)))) {
187         (void) fprintf(stderr,
188             "%s: value must be either 'true' or 'false': %s \n",
189             error_prefix, string);
190         return(FAILURE);
191     }
192     return(SUCCESS);
193 }
194
195 yes_or_no(string)
196     char *string;
197 {
198     if (strlen(string) == 0) {
199         return(FAILURE);
200     }
201     if ((0 != strncmp ("yes", string, strlen(string))) &&
202         (0 != strncmp("no", string, strlen(string)))) {
203         (void) fprintf(stderr,
204             "%s: value must be either 'yes' or 'no': %s \n",
205             error_prefix, string);
206         return(FAILURE);
207     }
208     return(SUCCESS);
209 }
210
211 lane_height_check (height)
212     char *height;
213 {
214     if (FAILURE == integer_check(height))
215         return(FAILURE);
216
217     if ((atoi(height)<=4) && (atoi(height)>0))
218         return(SUCCESS);
219
220     (void) fprintf(stderr,
221         "%s: Device adapter height may not be greater than 4 \n",
222         error_prefix);
223     return(FAILURE);
224 }
225
226 slot_width_check (width)
227     char *width;
228 {
229     if (FAILURE == integer_check(width))
230         return(FAILURE);
231
232     if ((atoi(width)<=8) && (atoi(width)>0))
233         return(SUCCESS);
234
235     (void) fprintf(stderr,

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
3/3

LINE #	SOURCE TEXT
241	"%s: device adapter width may not be greater than 8 \n",
242	error_prefix);
243	
244	return(FAILURE);
245	}
246	
247	port_check (string)
248	char *string;
249	{
250	{
251	if (strlen(string) == 0) {
252	return (FAILURE);
253	}
254	if (0 == str_cmp("console", string, strlen(string)))
255	return(SUCCESS);
256	if (0 == str_cmp("modem", string, strlen(string)))
257	return(SUCCESS);
258	(void) fprintf(stderr, "%s: invalid port name: '%s' \n",
259	error_prefix, string);
260	return(FAILURE);
261	}
262	
263	baud_rate_check (string)
264	char *string;
265	{
266	if (strlen(string) == 0) {
267	return (FAILURE);
268	}
269	if (0 == str_cmp("300", string))
270	return(SUCCESS);
271	if (0 == str_cmp("1200", string))
272	return(SUCCESS);
273	if (0 == str_cmp("2400", string))
274	return(SUCCESS);
275	if (0 == str_cmp("4800", string))
276	return(SUCCESS);
277	if (0 == str_cmp("9600", string))
278	return(SUCCESS);
279	if (0 == str_cmp("2400+", string))
280	return(SUCCESS);
281	if (0 == str_cmp("9600+", string))
282	return(SUCCESS);
283	(void) fprintf(stderr, "%s: invalid baud rate: '%s' \n",
284	error_prefix, string);
285	return(FAILURE);
286	}
287	
288	mode_check (mode)
289	char *mode;
290	{
291	if (strlen(mode) == 0) {
292	return(FAILURE);
293	}
294	if ((0 == str_cmp("diagnostics", mode, strlen(mode)))
295	(0 == str_cmp("normal", mode, strlen(mode)))) {
296	return(SUCCESS);
297	}
298	(void) fprintf(stderr,
299	"%s: mode must be either 'normal' or 'diagnostic' \n",
300	error_prefix);
301	return(FAILURE);
302	}
303	
304	minute_check (minute)
305	char *minute;
306	{
307	if (FAILURE == integer_check(minute))
308	return(FAILURE);
309	if ((atoi(minute)<1001) && (atoi(minute)>-1))
310	return(SUCCESS);
311	(void) fprintf(stderr, "%s: minutes must be between 0 and 1000\n",
312	error_prefix);
313	return(FAILURE);
314	}
315	
316	internet_port_check (port_number)
317	char *port_number;
318	{
319	if (FAILURE == integer_check(port_number))
320	return(FAILURE);
321	if ((atoi(port_number)<MIN_PORT) && (atoi(port_number)>MAX_PORT))
322	return(SUCCESS);
323	(void) fprintf(stderr,
324	"%s: internet port number must be between %d and %d\n",
325	error_prefix, MIN_PORT, MAX_PORT);
326	return(FAILURE);
327	}
328	
329	address_check (addr)
330	char *addr;
331	{
332	int loop;
333	}

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
4/4

LINE #	SOURCE TEXT
361	for (loop = 0; loop < strlen(addr); loop++) {
362	if (((addr[loop] != 'X') && (addr[loop] != 'x')) &&
363	(FAILURE == isdigit(addr[loop]))) {
364	(void) fprintf(stderr,
365	"%s: value must be numeric: %s \n",
366	error_prefix, addr);
367	return(FAILURE);
368	}
369	
370	if ((strlen(addr) > 10)
371	((strlen(addr) == 10) &&
372	(str_cmp(addr, "4294705156") > 0))) {
373	(void) fprintf(stderr, "%s: Address too large: %s \n",
374	error_prefix, addr);
375	return(FAILURE);
376	}
377	
378	return(SUCCESS);
379	}
380	
381	null_ok()
382	{
383	return(SUCCESS);
384	}
385	
386	always_ok(file_name)
387	char *file_name;
388	{
389	if (strlen(file_name) == 0) {
390	return(FAILURE);
391	}
392	return (SUCCESS);
393	}
394	
395	library_file_check(file_name)
396	char *file_name;
397	{
398	char ret_file[max_str];
399	if (strlen(file_name) == 0) {
400	return(FAILURE);
401	}
402	LM_CHECK(lm_resolve_library_file("LM_LIB", file_name, ret_file, max_str));
403	return (SUCCESS);
404	}
405	
406	file_check(file_name)
407	char *file_name;
408	{
409	int status;
410	if (strlen(file_name) == 0) {
411	return(FAILURE);
412	}
413	status = access(file_name, 0);
414	if (status != 0)
415	(void) fprintf(stderr,
416	"%s: file does not exist or cannot be accessed: %s \n",
417	error_prefix, file_name);
418	return (FAILURE);
419	return (SUCCESS);
420	}
421	
422	file_list_check (file_list)
423	char *file_list;
424	{
425	char *cur_file;
426	char current_file[max_str];
427	char delimiter[2];
428	char *name_pointer;
429	char *end_of_string;
430	long return_status, status;
431	if (strlen(file_list) == 0) {
432	return(FAILURE);
433	}
434	return_status = SUCCESS;
435	name_pointer = file_list;
436	/* find out if it's a comma or space separated list */
437	cur_file = (char *) strpbrk (file_list, " ,");
438	if (NULL == cur_file) cur_file = file_list + strlen(file_list);
439	(void) strcpy (delimiter, cur_file, 1);
440	delimiter[1] = NULL;
441	end_of_string = file_list + strlen(file_list);
442	/* check each file in the list */
443	do {
444	cur_file = (char *) strpbrk (name_pointer, delimiter);
445	if (NULL == cur_file) cur_file = file_list + strlen(file_list);
446	(void) strcpy (current_file, name_pointer, cur_file - name_pointer);
447	current_file[cur_file - name_pointer] = NULL;
448	clear_input(current_file);
449	name_pointer = cur_file + 1;
450	status = access(current_file, 0);
451	if (status != 0) {

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
5/5

```

LINE #          SOURCE TEXT
481             (void) fprintf(stderr,
482             "ts: File does not exist or cannot be accessed: ts \n",
483             error_prefix, current_file);
484             return_status = FAILURE;
485         }
486     }
487     while (cur_file != end_of_string),
488     return(return_status);
489
490 }
491
492 all_modeler_check(modeler_name)
493 char *modeler_name;
494 {
495     long number_of_modelers, modeler;
496     char **list_of_modelers;
497
498     LM_CHECK(lm_inquire_modeler_list(&number_of_modelers,
499     &list_of_modelers));
500
501     for (modeler=0; modeler<number_of_modelers;
502     modeler++, list_of_modelers++) {
503         if (0 == str_cmp(modeler_name, *list_of_modelers))
504             return(SUCCESS);
505     }
506
507     if (0 == str_cmp(modeler_name, "all"))
508         return(SUCCESS);
509
510     (void) fprintf(stderr, "ts: unknown modeler: ts \n",
511     error_prefix, modeler_name);
512
513     /*let user specify modeler not in modelers list*/
514     return(SUCCESS);
515 }
516
517 modeler_check(modeler_name)
518 char *modeler_name;
519 {
520     long number_of_modelers, modeler;
521     char **list_of_modelers;
522
523     if (strlen(modeler_name) == 0) return(FAILURE);
524
525     LM_CHECK(lm_inquire_modeler_list(&number_of_modelers,
526     &list_of_modelers));
527
528     for (modeler=0; modeler<number_of_modelers; modeler++,
529     list_of_modelers++) {
530         if (0 == str_cmp(modeler_name, *list_of_modelers))
531             return(SUCCESS);
532     }
533
534     (void) fprintf(stderr,
535     "ts: unknown modeler: \"%s\" \n", error_prefix, modeler_name);
536
537     /*let user specify modeler not in modelers list*/
538     return(SUCCESS);
539 }
540
541 mfg_check(mfg_name)
542 char *mfg_name;
543 {
544     if (strlen(mfg_name)>NAME_LENGTH) {
545         (void) fprintf(stderr,
546         "ts: manufacturer name must be %d characters or less: ts \n",
547         error_prefix, NAME_LENGTH, mfg_name);
548         return(FAILURE);
549     }
550     return(SUCCESS);
551 }
552
553 device_check(device_name)
554 char *device_name;
555 {
556     if (strlen(device_name)>NAME_LENGTH) {
557         (void) fprintf(stderr,
558         "ts: device name must be %d characters or less: ts \n",
559         error_prefix, NAME_LENGTH, device_name);
560         return(FAILURE);
561     }
562     return(SUCCESS);
563 }
564
565 adaptor_check(adaptor_type)
566 char *adaptor_type;
567 {
568     if (strlen(adaptor_type)>TYPE_LENGTH) {
569         (void) fprintf(stderr,
570         "ts: adaptor type must be %d characters or less: ts \n",
571         error_prefix, TYPE_LENGTH, adaptor_type);
572         return(FAILURE);
573     }
574     return(SUCCESS);
575 }
576
577 rev_check(rev_string)
578 char *rev_string;
579 {
580     if (strlen(rev_string)>REV_LENGTH) {
581         (void) fprintf(stderr,
582         "ts: revision string must be %d characters or less: ts \n",
583         error_prefix, REV_LENGTH, rev_string);
584         return(FAILURE);
585     }
586     return(SUCCESS);
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/checks.c

DATE 5/23/89
TIME 1:20:32 pm

PAGE #
6/6

LINE #	SOURCE TEXT
601	segment_check (segment_number)
602	char *segment_number,
603	int slot_number,
604	{
605	if (FAILURE == integer_check(segment_number))
606	return(FAILURE);
607	
608	slot_number = atoi(segment_number);
609	if ((slot_number < MIN_SEG) (slot_number > MAX_SEG)) {
610	(void) fprintf(stderr,
611	"%s: segment number must be between %d and %d: %s \n",
612	error_prefix, MIN_SEG, MAX_SEG, segment_number);
613	return(FAILURE);
614	}
615	
616	return (SUCCESS);
617	}
618	
619	slot_check (slot_string)
620	char *slot_string;
621	{
622	int slot_number;
623	
624	if (FAILURE == integer_check(slot_string))
625	return(FAILURE);
626	
627	slot_number = atoi(slot_string);
628	if ((slot_number < MIN_SLOT) (slot_number > MAX_SLOT)) {
629	(void) fprintf(stderr, "%s: invalid slot number: %s \n",
630	error_prefix, slot_string);
631	return(FAILURE);
632	}
633	
634	return(SUCCESS);
635	}
636	
637	lane_check (lane_id)
638	char *lane_id;
639	{
640	if (strlen(lane_id) == 0) {
641	return(FAILURE);
642	}
643	
644	if (strlen(lane_id) > 1) {
645	(void) fprintf(stderr, "%s: invalid lane: %s \n",
646	error_prefix, lane_id);
647	return(FAILURE);
648	}
649	
650	if (((lane_id[0] < MIN_LANE) (lane_id[0] > MAX_LANE))
651	((lane_id[0] < toupper(MIN_LANE))
652	(lane_id[0] > toupper(MAX_LANE)))) {
653	(void) fprintf(stderr, "%s: invalid lane: %s \n",
654	error_prefix, lane_id);
655	return(FAILURE);
656	}
657	
658	return(SUCCESS);
659	}
660	
661	lm_check (status)
662	int status;
663	{
664	char *message;
665	int severity;
666	
667	if (LM_SUCCESS != status) {
668	do {
669	lm_get_message(&severity, &message);
670	if (severity == LM_ERROR) {
671	(void) fprintf(stderr, "LM modeler error: %s\n", message);
672	}
673	
674	if (severity == LM_WARNING)
675	(void) fprintf(stderr, "LM modeler warning: %s\n",
676	message);
677	} while (severity != LM_NO_MORE_MESSAGES);
678	
679	return (status);
680	}
681	
682	
683	lm_diag_check (status)
684	int status;
685	{
686	char *message;
687	int severity;
688	
689	if (LM_SUCCESS != status) {
690	do {
691	lm_get_message(&severity, &message);
692	if (severity == LM_ERROR) {
693	(void) fprintf(stderr, "LM modeler error: %s\n",
694	message);
695	}
696	
697	} while (severity != LM_NO_MORE_MESSAGES);
698	
699	return (status);
700	}
701	

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

lm/command.c

DATE

5/23/89

PAGE #

TIME

1:20:33 pm

1/7

```

LINE # SOURCE TEXT
1  /* SCSS_ID: command.c rev 3.1.1, 4/24/89 at 07:51:41 */
2
3  /*
4  **
5  ** Text based utility
6  ** This module accepts and parses the commands entered
7  ** by the user
8  **
9  ** Copyright Logic Modeling Systems Incorporated, 1988
10 **
11 */
12 #include <stdio.h>
13 #include <ctype.h>
14 #include <sys/types.h>
15 #include <time.h>
16 #include <stdlib.h>
17 #include <string.h>
18 #include "common.h"
19 #include "lm_text.h"
20 #include "lm_rm_or.h"
21 #include "lm_rm.h"
22 #include "mod_err.h"
23 #include "version.h"
24 #include "short.h"
25 #include "names.h"
26 #include "ops.h"
27 #include "id.h"
28 #include "mod_def.h"
29 #include "network.h"
30 #include "command.h"
31
32 extern char error_prefix[];
33
34 int re_boot(), install(), shut_down(), diegs(), burnin(), qa(), boot();
35 int label(), set_password(),
36 play_vectors(), measure_timing(), check_shell_software(),
37 set_descriptors(), set_net_timeout(), show_modeler(),
38 show_logic_models(), show_users(), show_versions(), text_menu(),
39 null_ok(), always_ok(), lane_check(),
40 slot_check(), board_check(), lm_promote(),
41 file_list_check(), file_check(), library_file_check(), modeler_check(),
42 true_or_false(), yes_or_no(),
43 host_check(), test_check(), integer_check(), device_check(),
44 all_modeler_check(), address_check(), mode_check(),
45 minute_check(), lane_height_check(), mfg_check(),
46 internet_port_check(), adapter_check(), rev_check(),
47 segment_check(),
48 baud_rate_check(), port_check(), create_timing_file(),
49 command_help(), locate_model_file(),
50 abort_user(),
51 user_number_check();
52
53 struct param_st prompt_table[] = {
54 {PR_MODELER_NAME, "", "Modeler Name",
55  LM_FALSE, "Modeler Name",
56  {PR_MODELER_NAME, "all", "Modeler Name",
57  LM_FALSE, "Modeler Name",
58  {PR_LANE, "lane_check", "Lane (A - D)",
59  LM_FALSE, "Lane (A - D)",
60  {PR_SLOT, "slot_check", "Slot (0 - 7)",
61  LM_FALSE, "Slot (0 - 7)",
62  {PR_DEVICE_NAME, "device_check", "Device Name",
63  LM_FALSE, "Device Name",
64  {PR_MODEL_FILE, "library_file_check", "Model File (XXX.MDL)",
65  LM_FALSE, "Model File (XXX.MDL)",
66  {PR_PATTERN_FILE, "library_file_check", "Vector File (XXX.TST)",
67  LM_FALSE, "Vector File (XXX.TST)",
68  {PR_TIMING_FILE, "always_ok", "Timing File (XXX.TIM)",
69  LM_FALSE, "Timing File (XXX.TIM)",
70  {PR_OUTPUT_FILE, "null_ok", "Output File (XXX.OUT)",
71  LM_FALSE, "Output File (XXX.OUT)",
72  {PR_LOC_FILE, "null_ok", "Vector Log File (XXX.LOC)",
73  LM_FALSE, "Vector Log File (XXX.LOC)",
74  {PR_BOOT_LIST, "null_ok", "Boot List",
75  LM_FALSE, "Boot List",
76  {PR_FILENAME, "null_ok", "Application File",
77  LM_FALSE, "Application File",
78  {PR_START_ADDRESS, "address_check", "Start Address",
79  LM_FALSE, "Start Address",
80  {PR_MODE, "mode_check", "normal", "Mode",
81  LM_FALSE, "Mode",
82  {PR_WAIT, "yes_or_no", "no", "Wait",
83  LM_FALSE, "Wait",
84  {PR_MINUTES, "minute_check", "0", "Minutes",
85  LM_FALSE, "Minutes",
86  {PR_AUTOBOOT, "yes", "yes", "Autoboot",
87  LM_FALSE, "Autoboot",
88  {PR_HOST_NAME, "boot_check", "Boot Host Name",
89  LM_FALSE, "Boot Host Name",
90  {PR_DAB_MAKER, "mfg_check", "Model Developer",
91  LM_FALSE, "Model Developer",
92  {PR_DAB_MAKER_REV, "rev_check", "Model Revision",
93  LM_FALSE, "Model Revision",
94  {PR_MANUFACTURER, "mfg_check", "Manufacturer",
95  LM_FALSE, "Manufacturer",
96  {PR_REVISION, "rev_check", "Revision",
97  LM_FALSE, "Revision",
98  {PR_DAB_TYPE, "adapter_check", "Adapter Type",
99  LM_FALSE, "Adapter Type",
100 {PR_LANE_HIGH, "lane_height_check", "1", "Lanes High",
101 LM_FALSE, "Lanes High",
102 {PR_SLOTS_WIDE, "slot_check", "1", "Slots Wide",
103 LM_FALSE, "Slots Wide",
104 {PR_SEGMENT, "segment_check", "0", "Segment Number",
105 LM_FALSE, "Segment Number",
106 {PR_INET_PORT_NO, "internet_port_check", "1234", "Internet Port Number",
107 LM_FALSE, "Internet Port Number",
108 {PR_BAUD_RATE, "baud_rate_check", "9600", "Baud Rate",
109 LM_FALSE, "Baud Rate",
110 {PR_PORT, "port_check", "console", "Port",
111 LM_FALSE, "Port",
112 {PR_TIM_FILE, "file_list_check", "", "TIM File(s)",
113 LM_FALSE, "TIM File(s)",
114 {PR_TMG_FILE, "always_ok", "TMG File",
115 LM_FALSE, "TMG File",
116 {PR_FILE_NAME, "always_ok", "File Name",
117 LM_FALSE, "File Name",
118 {PR_BUS_MODE, "yes", "yes", "Combine Bus Delays",
119 LM_FALSE, "Combine Bus Delays",
120 {PR_USER_NO, "yes_or_no", "", "User Number",

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/command.c

DATE 5/23/89
TIME 1:20:33 pm

PAGE #
2/8

LINE #	SOURCE TEXT
121	user_number_check, IN_FALSE)
122	1,
123	
124	struct flag_at global_flag_table[] = {
125	{"model_name", 'm', PR_MODELER_NAME},
126	0
127	1,
128	
129	struct flag_at kill_ft[] = {
130	{"user_number", 'u', PR_USER_NO},
131	0
132	1,
133	
134	struct flag_at boot_ft[] = {
135	{"boot_list", 'b', PR_BOOT_LIST},
136	{"filename", 'f', PR_FILENAME},
137	{"start_address", 's', PR_START_ADDRESS},
138	{"mode", 'd', PR_MODE},
139	0
140	1,
141	
142	struct flag_at install_ft[] = {
143	{"autoboot", 'a', PR_AUTOBOOT},
144	{"boot_name", 'b', PR_BOOT_NAME},
145	{"internet_port_number", 'i', PR_INET_PORT_NO},
146	0
147	1,
148	
149	struct flag_at cr_tmg_file_ft[] = {
150	{"tmg_file", 't', PR_TMG_FILE},
151	{"tmg_file", 'g', PR_TMG_FILE},
152	{"bus_mode", 'u', PR_BUS_MODE},
153	0
154	1,
155	
156	struct flag_at reboot_ft[] = {
157	{"wait", 'w', PR_WAIT},
158	{"minutes", 'm', PR_MINUTES},
159	0
160	1,
161	
162	struct flag_at label_ft[] = {
163	{"lane", 'l', PR_LANE},
164	{"slot", 's', PR_SLOT},
165	{"device_name", 'd', PR_DEVICE_NAME},
166	{"segment", 'n', PR_SEGMENT},
167	{"manufacturer", 'p', PR_DAB_MAKER},
168	{"revision", 'r', PR_DAB_MAKER_REV},
169	{"manufacturer", 'f', PR_MANUFACTURER},
170	{"revision", 'r', PR_REVISION},
171	{"dab_type", 't', PR_DAB_TYPE},
172	{"lanes_high", 'h', PR_LANES_HIGH},
173	{"slots_wide", 'w', PR_SLOTS_WIDE},
174	0
175	1,
176	
177	struct flag_at play_vectors_ft[] = {
178	{"model_name", 'm', PR_MODEL_NAME},
179	{"pattern_file", 'p', PR_PATTERN_FILE},
180	{"output_file", 'o', PR_OUTPUT_FILE},
181	{"log_file", 'l', PR_LOG_FILE},
182	0
183	1,
184	
185	struct flag_at measure_timing_ft[] = {
186	{"model_name", 'm', PR_MODEL_NAME},
187	{"pattern_file", 'p', PR_PATTERN_FILE},
188	{"output_file", 'o', PR_OUTPUT_FILE},
189	{"timing_file", 't', PR_TIMING_FILE},
190	{"log_file", 'l', PR_LOG_FILE},
191	0
192	1,
193	
194	struct flag_at check_ft[] = {
195	{"model_name", 'm', PR_MODEL_NAME},
196	0
197	1,
198	
199	struct flag_at set_baud_ft[] = {
200	{"baud_rate", 'b', PR_BAUD_RATE},
201	{"port", 'p', PR_PORT},
202	0
203	1,
204	
205	struct flag_at set_net_ft[] = {
206	{"minutes", 'm', PR_MINUTES},
207	0
208	1,
209	
210	struct flag_at shutdown_ft[] = {
211	{"wait", 'w', PR_WAIT},
212	{"minutes", 'm', PR_MINUTES},
213	0
214	1,
215	
216	struct flag_at locate_ft[] = {
217	{"filename", 'f', PR_FILE_NAME},
218	0
219	1,
220	
221	struct command_at command_table [] = {
222	{"abort_user", abort_user, kill_ft, BMS},
223	{"boot", boot, boot_ft, BMS},
224	{"burnin", burnin, 0, BMS},
225	{"change_password", set_password, 0, BMS},
226	{"check", check_shell_software, check_ft, BMS},
227	{"create_timing", create_timing_file, cr_tmg_file_ft, BMS},
228	{"find", locate_model_file, locate_ft, NO_BMS},
229	{"diagnostics", diag, 0, BMS},
230	{"help", command_mode_help, 0, NO_BMS},
231	{"install", install, install_ft, BMS},
232	{"label", label, label_ft, BMS},
233	{"measure_timing", measure_timing, measure_timing_ft, BMS},
234	{"menu", text_menu, 0, BMS},
235	{"play_vectors", play_vectors, play_vectors_ft, BMS},
236	{"qa", qa, 0, BMS},
237	{"reboot", re_boot, reboot_ft, BMS},
238	{"set_baud", set_baudrate, set_baud_ft, BMS},
239	{"set_network", set_net_timeout, set_net_ft, BMS},
240	{"show logic models", show_logic_models, 0, BMS},

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/command.c

DATE 5/23/89 PAGE #
TIME 1:20:33 pm 3/9

```

LINE # SOURCE TEXT
241 {"show_modeler", show_modeler, 0, RMS},
242 {"show_users", show_users, 0, RMS},
243 {"show_versions", show_versions, 0, RMS},
244 {"shutdown", shut_down, shutdown_ft, RMS},
245 0
246 },
247
248 struct param_st *find_tag_param();
249 struct param_st *find_flag_param();
250 struct param_st *find_string_param();
251 struct command_st *find_command();
252
253 int command_mode = TRUE;
254 int prompt_mode = FALSE;
255
256 command_line_mode(argc, argv)
257 int argc;
258 char *argv[];
259
260 /* command line mode, looks and smells like UNIX (tm) */
261 {
262     int cur_arg;
263     struct command_st *command = 0;
264     struct flag_st *flag_st = 0;
265     struct param_st *param_st;
266     void find_and_execute();
267     jmp_buf jmpbuf;
268     extern jmp_buf *Jmpbuf;
269     #ifdef SUNOS3_5
270     int handle_intr();
271     #else
272     void handle_intr();
273     #endif
274
275     Jmpbuf = (jmp_buf *)Jmpbuf;
276     if (setjmp(Jmpbuf)) {
277         /* got an interrupt */
278         u_exit(); /* never returns */
279     }
280     (void) signal(SIGINT, handle_intr);
281
282     for (cur_arg = 1; cur_arg < argc; ++cur_arg) {
283         if (argv[cur_arg][0] == '-') {
284             if ((cur_arg == (argc - 1)) ||
285                 (argv[cur_arg + 1][0] == '-')) {
286                 (void) fprintf(stderr,
287                     "Missing parameter value for %s\n",
288                     argv[cur_arg]);
289                 u_exit(); /* never returns */
290             }
291             if (strlen(argv[cur_arg]) == 2) {
292                 param_st = find_flag_param(flag_st,
293                     argv[cur_arg][1]);
294             } else {
295                 param_st = find_string_param(flag_st,
296                     argv[cur_arg][1]);
297             }
298             if (param_st) {
299                 (void) strcpy(param_st->default_value,
300                     argv[cur_arg]);
301                 param_st->defined = LM_TRUE;
302             } else {
303                 (void) fprintf(stderr, "%s: unknown parameter: %s\n",
304                     error_prefix, argv[cur_arg]);
305                 u_exit();
306             }
307         } else if (!strcmp(argv[cur_arg], "prompt")) {
308             prompt_mode = TRUE;
309         } else if (!strcmp(argv[cur_arg], "field_engineering")) {
310             lm_promote();
311         } else {
312             /* must be a command */
313             if (command) {
314                 execute_command(command);
315             }
316             if (! (command = find_command(argv[cur_arg])) ) {
317                 (void) fprintf(stderr, "%s: unknown command: %s\n",
318                     error_prefix, argv[cur_arg]);
319                 u_exit(); /* never returns */
320             }
321             flag_st = command->flag_st;
322         }
323         if (command) execute_command(command);
324     }
325     find_and_execute("menu"); /* default command is "menu" */
326     u_exit(); /* never returns */
327 }
328
329 find_flag_tag(flag_st, c)
330 struct flag_st *flag_st;
331 char c;
332 {
333     /* look in local flag table if it exists */
334     if (flag_st) for (flag_st->tag, ++flag_st) {
335         if (flag_st->flag == c) {
336             return flag_st->tag;
337         }
338     }
339     /* look in global flag table */
340     for (flag_st = global_flag_table; flag_st->tag, ++flag_st) {
341         if (flag_st->flag == c) {
342             return flag_st->tag;
343         }
344     }
345     return 0; /* undefined flag */
346 }
347
348 find_string_tag(flag_st, s)
349 struct flag_st *flag_st;
350 char *s;
351 {

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/command.c

DATE 5/23/89
TIME 1:20:33 pm

PAGE #
4/10

LINE #	SOURCE TEXT
361	{
362	/* look in local flag table if it exists */
363	if (flag_st) for (; flag_st->tag; ++flag_st) {
364	if (str_cmp(s, flag_st->string) == 0) {
365	return flag_st->tag;
366	}
367	}
368	
369	/* look in global flag table */
370	for (flag_st = global_flag_table; flag_st->tag; ++flag_st) {
371	if (str_cmp(s, flag_st->string) == 0) {
372	return flag_st->tag;
373	}
374	}
375	
376	return 0; /* undefined flag */
377	}
378	
379	
380	
381	struct param_st *
382	find_tag_param(tag)
383	{
384	struct param_st *param_st;
385	
386	for (param_st = prompt_table; param_st->tag; ++param_st) {
387	if (param_st->tag == tag) {
388	return param_st;
389	}
390	}
391	
392	return 0; /* undefined tag */
393	}
394	
395	
396	
397	struct param_st *
398	find_flag_param(flag_st, c)
399	{
400	struct flag_st *flag_st;
401	char c;
402	{
403	return find_tag_param(find_flag_tag(flag_st, c));
404	}
405	
406	
407	
408	struct param_st *
409	find_string_param(flag_st, s)
410	{
411	struct flag_st *flag_st;
412	char *s;
413	{
414	return find_tag_param(find_string_tag(flag_st, s));
415	}
416	
417	
418	
419	struct command_st *
420	find_command(s)
421	{
422	struct command_st *command;
423	{
424	for (command = command_table; command->command; ++command) {
425	if (strncmp(command->command, s, strlen(s)) == 0) {
426	return command;
427	}
428	}
429	return 0;
430	}
431	
432	
433	execute_command(command)
434	{
435	struct command_st *command;
436	{
437	if (command->hms == HMS) {
438	local_begin_modeling_session();
439	}
440	(*command->routine)();
441	local_end_modeling_session();
442	}
443	
444	void
445	find_and_execute(s)
446	{
447	char *s;
448	{
449	struct command_st *command;
450	
451	if (command = find_command(s))
452	execute_command(command);
453	else
454	(void) fprintf(stderr, "%s: unknown command: %s\n", error_prefix, s);
455	}
456	
457	command_mode_help()
458	{
459	struct command_st *command;
460	struct flag_st *flag_st;
461	
462	(void) printf("Command mode command summary:\n");
463	
464	for (flag_st = global_flag_table; flag_st->flag; ++flag_st) {
465	(void) printf("\t

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/command.c

DATE 5/23/89
TIME 1:20:33 pm

PAGE #
5/11

```

LINE #          SOURCE TEXT
481      int      tag;
482      int      (*routine)();
483
484      struct param_st *param_st = find_tag_param(tag);
485
486      if (param_st) {
487          param_st->validate = routine;
488          return(SUCCESS);
489      }
490      (void) fprintf(stderr,
491          "ls: internal error 1: bad parameter tag value: %d\n",
492          error_prefix, tag);
493      return(FAILURE);
494 }
495
496
497
498 set_default(tag, default_value)
499     int      tag;
500     char      *default_value;
501
502     struct param_st *param_st = find_tag_param(tag);
503
504     if (param_st) {
505         param_st->defined = LM_TRUE;
506         (void) strcpy(param_st->default_value, default_value);
507         return(SUCCESS);
508     }
509     (void) fprintf(stderr,
510         "ls: internal error 1: bad parameter tag value: %d\n",
511         error_prefix, tag);
512     return(FAILURE);
513 }
514
515
516
517 char *
518 get_default(tag)
519     int      tag;
520
521     struct param_st *param_st;
522
523     if (param_st = find_tag_param(tag)) {
524         return param_st->default_value;
525     }
526
527     return 0; /* undefined tag */
528 }
529
530
531
532 get_parameter (tag, return_value, size)
533     int      tag;
534     char      *return_value;
535     int      size;
536
537     char      reply[max_str];
538     char      *reply_ptr;
539     char      prompt[max_str];
540     int      number_of_modelers;
541     char      *list_of_modelers;
542     struct param_st *param_st;
543
544     if (!(param_st = find_tag_param(tag))) {
545         (void) fprintf(stderr,
546             "ls: internal error 2: bad parameter tag value: %d\n",
547             error_prefix, tag);
548         return(FAILURE);
549     }
550
551     /* special handling for get_modeler_name */
552     /* if there's only one modeler, don't ask */
553
554     switch (tag) {
555
556     case PR_MODELER_DISP:
557     case PR_MODELER_NAME:
558
559         (void) lm_check(lm_inquire_modeler_list(&number_of_modelers,
560             &list_of_modelers));
561
562         if (number_of_modelers == 1) {
563             reply_ptr = get_default (PR_MODELER_NAME);
564             if ((strlen(reply_ptr) == 0) || (0 == str_cmp(reply_ptr, "list_of_modelers"))) {
565                 (void) strcpy (return_value, "list_of_modelers");
566                 return (SUCCESS);
567             }
568         }
569     }
570
571     (void) sprintf(prompt, (strlen(param_st->default_value) != 0)?
572         "ls [%s]: " : "ls: ", param_st->prompt_string,
573         param_st->default_value);
574
575     if (command_mode == TRUE) {
576         (void) strcpy (return_value, param_st->default_value);
577         if (((! param_st->validate)
578             (return_value)) == SUCCESS) return SUCCESS;
579
580         if ((prompt_mode == FALSE) && (!(is_param_defined(tag)))
581             (void) fprintf(stderr, "ls not specified\n", param_st->prompt_string);
582
583         return FAILURE;
584     }
585
586     /* prompt for value */
587     do {
588         lm_get_input(prompt, reply, max_str);
589         if (strlen(reply) == 0)
590             (void)strcpy (reply, param_st->default_value);
591     } while (SUCCESS != (param_st->validate)(reply));
592
593     (void) strcpy(return_value, reply, size);
594     return(SUCCESS);
595 }
596
597
598
599 is_param_defined(tag)
600     int      tag;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/command.c

DATE 5/23/89
TIME 1:20:33 pm

PAGE #
6/12

LINE #	SOURCE TEXT
601	
602	{ struct param_st *param_st;
603	
604	if (param_st = find_tag_param(tag)) {
605	return param_st->defined;
606	}
607	
608	return 0; /* undefined tag */
609	}
610	
611	
612	
613	define_param(tag, state)
614	int tag;
615	long state;
616	
617	{ struct param_st *param_st;
618	
619	if (param_st = find_tag_param(tag)) {
620	param_st->defined = state;
621	}
622	}
623	
624	
625	
626	
627	
628	
629	
630	
631	
632	
633	
634	
635	

Copyright 1989
Logic Modeling Systems

HEADER FILE
lm/command.h

DATE 5/23/89
TIME 1:20:33 pm

PAGE #
1/13

```

LINE #      HEADER TEXT
1  /* SCCS_ID: command.h rev 3.1, 4/24/89 at 07:51:45 */
2
3  struct flag_st {
4      char    *string;
5      char    flag;
6      int     tag; /* tag in prompt_table */
7  };
8
9  struct param_st {
10     int     tag;
11     char    default_value[max_str];
12     char    *prompt_string;
13     int     (*validate)();
14     int     defined;
15 };
16
17 struct command_st {
18     char    *command;
19     int     (*routine)();
20     struct flag_st *flag_st;
21     int     hms; /* begin modeling session flag */
22 };
23
24 /* begin modeling session flag defines */
25
26 #define NO_HMS 0 /* don't begin modeling session before executing */
27 #define HMS 1 /* begin modeling session before executing */
28
29 /* Parameter table tag defines */
30
31 #define PR_MODELER_NAME 1
32 #define PR_MODELER_DISP 2
33 #define PR_LANE 3
34 #define PR_SLOT 4
35 #define PR_DEVICE_NAME 5
36 #define PR_MODEL_NAME 6
37 #define PR_PATTERN_FILE 7
38 #define PR_TIMING_FILE 8
39 #define PR_OUTPUT_FILE 9
40 #define PR_BOOT_LIST 10
41 #define PR_FILENAME 11
42 #define PR_START_ADDRESS 12
43 #define PR_MODE 13
44 #define PR_WAIT 14
45 #define PR_MINUTES 15
46 #define PR_AUTOBOOT 16
47 #define PR_HOST NAME 17
48 #define PR_MANUFACTURER 18
49 #define PR_DAB_TYPE 19
50 #define PR_REVISION 20
51 #define PR_LANES_WIDE 21
52 #define PR_SLOTS_WIDE 22
53 #define PR_SEGMENT 23
54 #define PR_INET_PORT_NO 24
55 #define PR_BAUD_RATE 25
56 #define PR_PORT 26
57 #define PR_TIM_FILE 27
58 #define PR_TWC_FILE 28
59 #define PR_BUS_MODE 29
60 #define PR_LOC_FILE 30
61 #define PR_USER_NO 31
62 #define PR_DAB_MARKER 32
63 #define PR_DAB_MARKER_REV 33
64 #define PR_FILE_NAME 34
65
66 struct command_st *find_command();
67 struct flag_st *find_param_string();
68 struct flag_st *find_param_flag();
69 int set_default();
70 char *get_default();

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
1/14

```

LINE # SOURCE TEXT
1  /* SCCS ID: display.c rev 2.10, 2/7/89 at 15:44:16 */
2  #include <stdio.h>
3  #include <ctype.h>
4  #include <sys/types.h>
5  #include <netinet/in.h>
6  #include <netdb.h>
7
8  #ifdef VMS
9  #include <descip.h>
10 #endif
11
12 #include "common.h"
13 #include "network.h"
14 #include "lm_text.h"
15 #include "lm_xd_wr.h"
16 #include "lm_xf.h"
17 #include "cpu.h"
18 #include "id.h"
19 #include "mod_def.h"
20 #include "mod_err.h"
21 #include "command.h"
22 #include "swaram.h"
23
24 char dash[] = "- ";
25 char blank[] = " ";
26
27 /* Kinda reminds you of Fortran, huh? print with 100 */
28
29 char fmt1[] = "%-32s%32s%32s%32s\n";
30 char fmt2[] = "%-32s%32s%32s%32s\n";
31 char fmt3[] = "%-32s%32s%32s%32s\n";
32 char fmt4[] = "%-20s%-12s%-7s%-13s%-8s%-6s%-4s\n";
33 char fmt5[] = "%-20s%-12s%-7s%32s / %-5d%-8s%-6s%-4s\n";
34 char fmt6[] = "%-10s%-10s%-6s%-8s%-14s%-14s%-14s%-14s\n";
35 char fmt7[] = "%-10s%-10s%32s%32s%32s / %-5d%-6s / %-6s%32s / %32s\n";
36
37 typedef char printline[81];
38
39 extern void read_every_forth_byte();
40 extern char *create_revision();
41
42 #ifdef VMS
43 extern vms_str_cmp();
44 #endif
45
46 extern int str_cmp(), str_bcmp(), /* case insensitive string compares */
47 extern char *string_it();
48 extern char error_prefix[max_str];
49 extern char *boot_host();
50 extern char *string_interest();
51 extern char *extract_version();
52
53 display_users(moduler_name)
54 char *moduler_name;
55 {
56     long user, number_of_users, pub_parts, priv_parts, pat_len, active_instances, idle_time,
57     long active_faults, pat_alloc, patstat;
58     long *list_of_users;
59     long time_status;
60     char **list_of_user_names, **list_of_host_names;
61     char b[10], c[10];
62     char idle_str[10];
63     char user_name[80], host_name[80];
64     printline *user_array;
65
66     LM_CHECK(lm_select_moduler(moduler_name));
67
68     /* get the total number of users on the system */
69
70     LM_CHECK(lm_inquire_user_list(&number_of_users, &list_of_users,
71     &list_of_host_names, &list_of_user_names));
72
73     /* warning: the user may logout while this function is running */
74
75     (void) printf("\n");
76     (void) printf("%s\n", "LM-1000 Users", moduler_name);
77     (void) printf("\n");
78     (void) printf(fmt4, "User", "Host", "User", "Idle", "Devices", "Patterns", "Simulator");
79     (void) printf(fmt4, "Name", "Name", "Number", "Time", "Public/Private", "Used/Allocated", "Instances");
80     (void) printf("\n");
81
82     user_array = (printline *) malloc ((unsigned int) number_of_users * sizeof(printline));
83
84     for (user = 0; user < number_of_users; user++, list_of_users++) {
85         /* truncate extra-long host and user names */
86         (void) strncpy (user_name, *list_of_user_names++, 10);
87         user_name[10] = '\0';
88         (void) strncpy (host_name, *list_of_host_names++, 10);
89         host_name[10] = '\0';
90
91         time_status = lm_inquire_user(*list_of_users,
92         LM_SECONDS SINCE LAST RESPONSE, &idle_time);
93
94         LM_CHECK_TO(dsp_user_cleanup,
95         lm_inquire_user(*list_of_users,
96         LM_NUMBER OF PUBLIC DEVICES, &pub_parts));
97         LM_CHECK_TO(dsp_user_cleanup,
98         lm_inquire_user(*list_of_users,
99         LM_NUMBER OF PRIVATE DEVICES, &priv_parts));
100        LM_CHECK_TO(dsp_user_cleanup,
101        lm_inquire_user(*list_of_users,
102        LM_NUMBER OF PATTERNS, &pat_len));
103
104        patstat = lm_inquire_user(*list_of_users, LM_NUMBER OF PATTERNS ALLOCATED, &pat_alloc);
105        if (LM_SUCCESS != patstat) pat_alloc = 0;
106
107        LM_CHECK_TO(dsp_user_cleanup,
108        lm_inquire_user(*list_of_users,
109        LM_NUMBER OF ACTIVE INSTANCES, &active_instances));
110        LM_CHECK_TO(dsp_user_cleanup,
111        lm_inquire_user(*list_of_users,
112        LM_NUMBER OF ACTIVE FAULTS, &active_faults));
113
114        if (time_status != LM_SUCCESS) {
115            (void) sprintf(idle_str, "----");
116        } else {
117            int days, hours, minutes, seconds;
118
119
120

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89

PAGE #

TIME 1:20:34 pm

2/15

```

LINE #          SOURCE TEXT
121      days = idle_time / (24 * 60 * 60); idle_time = idle_time % (24 * 60 * 60);
122      hours = idle_time / (60 * 60); idle_time = idle_time % (60 * 60);
123      minutes = idle_time / (60); idle_time = idle_time % (60);
124      seconds = idle_time;
125
126      if (days > 3) {
127          if (12 < hours) days += 1;
128          (void) sprintf(idle_str, "%3d days", days);
129      } else if ((0 == days) && (0 == hours)) {
130          (void) sprintf(idle_str, " %3d:%02d", minutes, seconds);
131      } else {
132          hours = hours + days * 24;
133          (void) sprintf(idle_str, "%2d:%02d:%02d", hours, minutes, seconds);
134      }
135
136      (void) sprintf((char *) (user_array + user), fmt7,
137                    user_name, host_name,
138                    *list_of_users, idle_str,
139                    pub_parts, priv_parts,
140                    string_it(pat_len, b, 10),
141                    string_it(pat_alloc, c, 10),
142                    active_instances,
143                    active_faults);
144      }
145
146      #ifdef VMS
147      qsort((char *) user_array, (int) number_of_users, sizeof(printline), str_cmp);
148      #else
149      vms_qsort((char *) user_array, (int) number_of_users, sizeof(printline), vms_str_cmp);
150      #endif
151
152      for (user = 0; user < number_of_users; user++) {
153          (void) printf("%s", user_array + user);
154      }
155
156      (void) printf("\n");
157
158      dep_user_cleanup:
159      free((char *) user_array);
160      return (SUCCESS);
161
162      }
163
164      show_logic_models()
165      {
166          char modeler_name[max_str];
167          long status;
168
169          status = get_parameter(PR_MODELER_NAME,
170                                modeler_name, sizeof(modeler_name));
171          if (status != SUCCESS)
172              return (FAILURE);
173          LM_CHECK(lm_select_modeler(modeler_name));
174          (void) display_logic_models(modeler_name);
175          return (SUCCESS);
176      }
177
178      show_users()
179      {
180          char modeler_name[max_str];
181          long status;
182
183          status = get_parameter(PR_MODELER_NAME,
184                                modeler_name, sizeof(modeler_name));
185          if (status != SUCCESS)
186              return (FAILURE);
187          LM_CHECK(lm_select_modeler(modeler_name));
188          (void) display_users(modeler_name);
189          return (SUCCESS);
190      }
191
192      show_modeler()
193      {
194          char modeler_name[max_str];
195          long status;
196
197          status = get_parameter(PR_MODELER_NAME,
198                                modeler_name, sizeof(modeler_name));
199          if (status != SUCCESS)
200              return (FAILURE);
201          LM_CHECK(lm_select_modeler(modeler_name));
202          (void) display_modeler(modeler_name);
203          return (SUCCESS);
204      }
205
206      show_versions()
207      {
208          char modeler_name[max_str];
209          long status;
210
211          status = get_parameter(PR_MODELER_NAME,
212                                modeler_name, sizeof(modeler_name));
213          if (status != SUCCESS)
214              return (FAILURE);
215          LM_CHECK(lm_select_modeler(modeler_name));
216          (void) display_versions(modeler_name);
217          return (SUCCESS);
218      }
219
220      }
221
222      }
223
224      }
225
226      }
227
228      }
229
230      }
231
232      }
233
234      }
235
236      }
237
238      }
239
240      }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
3/16

```

LINE #          SOURCE TEXT
241
242
243 show_per_modeler(display_function)
244 long (*display_function) ();
245
246 {
247     char selected_modeler(max_str), *modeler_name;
248     long status;
249     int number_of_modelers, modeler;
250     char *list_of_modelers;
251     char bogus(10);
252
253     LM_CHECK(lm_inquire_modeler_list(&number_of_modelers, &list_of_modelers));
254
255     status = get_parameter(PR_MODELER_DISP,
256                          selected_modeler, sizeof(selected_modeler));
257     if (status != SUCCESS)
258         return (FAILURE);
259
260     (void) printf("\n");
261
262     if (0 == str_cmp(selected_modeler, "all")) {
263         if (number_of_modelers == 0) {
264             (void) printf("No modelers are currently available\n");
265             return SUCCESS;
266         }
267         for (modeler = 0; modeler < (number_of_modelers - 1); modeler++) {
268             status = do_show_per_modeler(display_function,
269                                         "(list_of_modelers + modeler)");
270             if (status != LM_ERROR) {
271                 (void) lm_get_input("Press return for next modeler... ",
272                                     bogus, 10);
273                 (void) printf("\n");
274             }
275         }
276         modeler_name = "(list_of_modelers + modeler)";
277     } else {
278         modeler_name = selected_modeler;
279     }
280
281     /* last modeler in the list */
282
283     status = do_show_per_modeler(display_function, modeler_name);
284
285     if (status != LM_ERROR) {
286         (void) lm_get_input("Press return to continue... ", bogus, 10);
287         (void) printf("\n");
288     }
289     return (SUCCESS);
290 }
291
292
293 do_show_per_modeler(display_function, modeler_name)
294 long (*display_function) ();
295 char *modeler_name;
296 {
297     long status;
298     int severity;
299     char *message;
300
301     status = lm_select_modeler(modeler_name);
302     if (status == LM_ERROR) {
303         (void) printf("%s: Modeler \"%s\" unavailable: ", error_prefix, modeler_name);
304         {
305             lm_get_message(&severity, &message);
306             if ((severity == LM_ERROR) || (severity == LM_WARNING)) {
307                 (void) printf("%s\n", message);
308             }
309         }
310         while (severity != LM_NO_MORE_MESSAGES);
311     } else {
312         (*display_function) (modeler_name);
313     }
314
315     return (status);
316 }
317
318
319 display_modeler(modeler_name)
320 char *modeler_name;
321 {
322     char buffer(128);
323     long total_modeler_memory, total_pattern_memory, memory_used;
324     long avail_modeler_memory, avail_pattern_memory;
325     long number_of_users;
326     long number_of_lanes, number_of_slots;
327     long number_of_pacs, number_of_pams;
328     long number_of_pels, number_of_dabs, active_instances;
329     long active_faults, rev_no;
330     long status;
331     char b1(10), b2(10), b3(10), b4(10), b5(10);
332     ID_FROM_CPU cpu_id_struct;
333     BOOT_STRUCT boot_save;
334
335     LM_CHECK(lm_inquire_modeler
336              (LM_TOTAL_MODELER_MEMORY, &total_modeler_memory));
337     LM_CHECK(lm_inquire_modeler
338              (LM_TOTAL_PATTERNS, &total_pattern_memory));
339     LM_CHECK(lm_inquire_modeler
340              (LM_AVAILABLE_MODELER_MEMORY, &avail_modeler_memory));
341     LM_CHECK(lm_inquire_modeler
342              (LM_AVAILABLE_PATTERNS, &avail_pattern_memory));
343     LM_CHECK(lm_inquire_modeler
344              (LM_NUMBER_OF_USERS, &number_of_users));
345     LM_CHECK(lm_inquire_modeler
346              (LM_NUMBER_OF_LANES, &number_of_lanes));
347     LM_CHECK(lm_inquire_modeler
348              (LM_NUMBER_OF_SLOTS_PER_LANE, &number_of_slots));
349     LM_CHECK(lm_inquire_modeler
350              (LM_NUMBER_OF_PACS, &number_of_pacs));
351     LM_CHECK(lm_inquire_modeler
352              (LM_NUMBER_OF_PAMS, &number_of_pams));
353     LM_CHECK(lm_inquire_modeler
354              (LM_NUMBER_OF_PELS, &number_of_pels));
355     LM_CHECK(lm_inquire_modeler
356              (LM_NUMBER_OF_DABS, &number_of_dabs));
357     LM_CHECK(lm_inquire_modeler

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
4/17

```

LINE # SOURCE TEXT
361 (LM_NUMBER_OF_ACTIVE_INSTANCES, &active_instances));
362 LM_CHECK(lm_inquire_modeler
363 (LM_NUMBER_OF_ACTIVE_FAULTS, &active_faults));
364 LM_CHECK(lm_inquire_modeler
365 (LM_SOFTWARE_REVISION_NUMBER, &rev_no));
366
367 LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,
368 (u_long) 0, (u_long) 0, (u_long) sizeof(BOOT_STRUCT),
369 (char *) &boot_nvr. astatus));
370
371 read_every_fourth_byte(modeler_name, (u_long) CPU_ID_FROM,
372 (char *) &cpu_id_struct, (u_long) sizeof(cpu_id_struct));
373
374 memory_used = total_modeler_memory - avail_modeler_memory;
375
376 total_modeler_memory = (total_modeler_memory + (MEGABYTE / 2)) / MEGABYTE;
377 avail_modeler_memory = (avail_modeler_memory + (MEGABYTE / 2)) / MEGABYTE;
378
379 (void) printf("\n");
380 (void) printf("LM-1000 Configuration \n", modeler_name);
381 (void) printf("\n");
382 (void) printf("Modeler", "Quantity", "Lane A", "Lane B", "Lane C", "Lane D");
383 (void) printf("\n");
384 (void) printf("CPU", "I ", dash, dash, dash, dash);
385 (void) printf("CPU Memory Used (1GB total)", total_modeler_memory);
386 (void) printf("Pattern Controller", string_it(memory_used, bl, 6), dash, dash, dash, dash);
387 (void) printf("Timing Controller", "I ", dash, dash, dash, dash);
388
389 /* Determine number of PAMS per lane */
390 {
391     long pac[4], small_lane_pams[4], medium_lane_pams[4], lane_mem[4], lane_avail[4], used[4];
392     long i, j, size, has_a_pac, small_pams, medium_pams, num_pams;
393
394     size = 0;
395     has_a_pac = 0;
396     small_pams = 0;
397     medium_pams = 0;
398     num_pams = 0;
399
400     for (i = 0; i < number_of_lanes; i++)
401     {
402         pac[i] = small_lane_pams[i] = medium_lane_pams[i] =
403             lane_mem[i] = lane_avail[i] = used[i] = 0;
404     }
405
406     for (i = 0; i < number_of_lanes; i++) {
407         LM_CHECK(lm_inquire_lane(i, LM_IS_LANE_USABLE, &has_a_pac));
408         if (LM_TRUE == has_a_pac) {
409             pac[i] = 1;
410             LM_CHECK(lm_inquire_lane(i, LM_NUMBER_OF_PAMS, &num_pams));
411             for (j = 0; j < num_pams; j++) {
412                 LM_CHECK(lm_inquire_pam(i, j, LM_PAM_MEMORY_SIZE, &size));
413                 if ((128 * 1024) == size) {
414                     small_pams += size;
415                     small_lane_pams[i]++;
416                 }
417                 if ((512 * 1024) == size) {
418                     medium_pams += size;
419                     medium_lane_pams[i]++;
420                 }
421             }
422             LM_CHECK(lm_inquire_lane(i, LM_AVAILABLE_PATTERNS, &(lane_avail[i])));
423             LM_CHECK(lm_inquire_lane(i, LM_TOTAL_PATTERNS, &(lane_mem[i])));
424             lane_avail[i] += 256;
425             avail_patterns_memory += 256;
426             used[i] = lane_mem[i] - lane_avail[i];
427         }
428     }
429
430 (void) printf("Pattern Controller", number_of_pacs, pac[0], pac[1], pac[2], pac[3]);
431 (void) printf("\n");
432 (void) printf("128K Fast Pattern Memory");
433 string_it(small_pams, bl, 6), small_lane_pams[0], small_lane_pams[1],
434 small_lane_pams[2], small_lane_pams[3];
435 (void) printf("512K Fast Pattern Memory");
436 string_it(medium_pams, bl, 6), medium_lane_pams[0], medium_lane_pams[1],
437 medium_lane_pams[2], medium_lane_pams[3];
438 (void) printf("Total Pattern Memory");
439 string_it(total_patterns_memory, bl, 6),
440 string_it(lane_mem[0], b2, 6),
441 string_it(lane_mem[1], b3, 6),
442 string_it(lane_mem[2], b4, 6),
443 string_it(lane_mem[3], b5, 6));
444
445 (void) printf("\n");
446 (void) printf("Pattern Memory Available",
447 string_it(avail_patterns_memory, bl, 6),
448 string_it(lane_avail[0], b2, 6),
449 string_it(lane_avail[1], b3, 6),
450 string_it(lane_avail[2], b4, 6),
451 string_it(lane_avail[3], b5, 6));
452 (void) printf("Pattern Memory Allocated", blank,
453 string_it(used[0], bl, 6), string_it(used[1], b2, 6),
454 string_it(used[2], b3, 6), string_it(used[3], b4, 6));
455
456 }
457
458 long instances[4], faults[4], pels[4];
459 long i, has_a_pac;
460
461 for (i = 0; i < number_of_lanes; i++) {
462     instances[i] = faults[i] = pels[i] = 0;
463 }
464
465 for (i = 0; i < number_of_lanes; i++) {
466     LM_CHECK(lm_inquire_lane(i, LM_IS_LANE_USABLE, &has_a_pac));
467     if (LM_TRUE == has_a_pac) {
468         LM_CHECK(lm_inquire_lane(i, LM_NUMBER_OF_PELS, &(pels[i])));
469     }
470 }
471
472 (void) instances_per_lane(instances, faults);
473 (void) printf("\n");
474 (void) printf("Number of Simulator Instances", blank,
475 instances[0], instances[1], instances[2], instances[3]);
476 (void) printf("Number of Simulator Faults", blank,
477 faults[0], faults[1], faults[2], faults[3]);
478 (void) printf("Pin Electronics Modules",
479 number_of_pels, pels[0], pels[1], pels[2], pels[3]);
480 (void) printf("\n");

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
5/18

```

LINE #          SOURCE TEXT
481      char **list_of_host_names, **list_of_user_names;
482      long number_of_users;
483      long *list_of_users;
484      long i, real_users;
485
486      real_users = 0;
487
488      (void) printf(" Active Users: ");
489
490      LM_CHECK(lm_inquire_user_list(&number_of_users, &list_of_users,
491                                  &list_of_host_names, &list_of_user_names));
492
493      for (i = 0; i < number_of_users; i++, list_of_host_names++, list_of_user_names++) {
494          if ((strlen(list_of_user_names) > 0) && (strlen(list_of_host_names) > 0)) {
495              (void) printf("%s\n", *list_of_user_names, *list_of_host_names);
496              real_users++;
497              if (0 == ((real_users) & 4))
498                  (void) printf("\n");
499          }
500      }
501
502      if (0 != ((real_users) & 4))
503          (void) printf("\n");
504
505
506      {
507          extern char *lmsi_version;
508          char modeler_string[100];
509          long status;
510          unsigned long version_address;
511
512          status = lm_inquire_modeler(LM_GET_VERSION_STRING_ADDR, &version_address);
513          if (status == LM_SUCCESS) {
514              (void) lm_check(lm_read(modeler_name, (u_long) LM_MODELER_MEMORY,
515                                   (u_long) 0, (u_long) version_address, (u_long) 100,
516                                   (char *) modeler_string, &status));
517              (char *) modeler_string, &status);
518              modeler_string[99] = '\0'; // Make sure you null terminate string, just in case we read trash!
519          } else { strcpy(modeler_string, ""); }
520
521          (void) printf("\n");
522          (void) printf(" LM-1000 Core Modeler Code Version: %s\n", extract_version(modeler_string));
523          (void) printf(" LM-1000 Host Resident Code Version: %s\n", extract_version(lmsi_version));
524      }
525
526      if (boot_save.autoboot) {
527          char buffer[1024];
528
529          (void) printf(" Modeler is configured to autoboot from %s (%s)\n",
530                      boot_host(boot_save.host_internet_address),
531                      string_internet(boot_save.host_internet_address, buffer));
532      } else {
533          (void) printf(" Modeler is configured for manual boot\n");
534      }
535
536      (void) printf("\n");
537      return (SUCCESS);
538
539
540
541
542
543      char *
544      string_internet(internet_address, buffer)
545      unsigned long internet_address;
546      char *buffer;
547
548      {
549          char *string_internet_address;
550
551          internet_address = htonl(internet_address);
552
553          string_internet_address = buffer;
554
555          (void) sprintf(string_internet_address, "%d.%d.%d.%d",
556                      ((0xFF000000 & internet_address) >> 24),
557                      ((0x00FF0000 & internet_address) >> 16),
558                      ((0x0000FF00 & internet_address) >> 8),
559                      ((0x000000FF & internet_address) >> 0));
560          return (string_internet_address);
561      }
562
563
564      char *
565      extract_version(lmsi_version)
566      char *lmsi_version;
567
568      {
569          char *s, *t;
570
571          if ((NULL == lmsi_version) || (0 == strlen(lmsi_version)))
572              return ("PRE-RI.0");
573
574          if (0 == (s = strchr(lmsi_version, '.')))
575              return ("unknown version");
576
577          /* remove all non-printing characters (there shouldn't be, but just in case we got trash from the modeler) */
578          t = s;
579          while (*t) {
580              if (!isprint(*t))
581                  *t = '.';
582              t++;
583          }
584          return (*t);
585      }
586
587
588
589      char *
590      boot_host(internet_address)
591      unsigned long internet_address;
592
593      {
594          struct hostent *host_entry;
595
596          host_entry = gethostbyaddr((char *) &internet_address, 4, AF_INET);
597          if (NULL == host_entry) {
598              return("Unknown host");
599          }
600          return (host_entry->h_name);

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89

PAGE #

TIME 1:20:34 pm

6/19

```

LINE #          SOURCE TEXT
601
602 }
603
604
605
606 void
607 read_every_forth_byte(modeler_name, address, buffer, size)
608     char *modeler_name;
609     u_long address;
610     char *buffer;
611     u_long size;
612
613 {
614     int i;
615     long status;
616
617     for (i = 0; i < size; i++) {
618         (void) lm_check(lm_read(modeler_name, (u_long) LM_MODELER_MEMORY,
619             (u_long) 0, (u_long) address, (u_long) 1,
620             (char *) buffer + i, &status));
621         address += 4;
622     }
623 }
624
625
626 char *
627 string_it(value, buffer, size)
628     long value;
629     char *buffer;
630     short size;
631
632 {
633     long n;
634     static char suffix[] = " KMGTP";
635     char temp_buffer[max_str];
636
637     n = 0;
638
639     if (value > 1023) {
640         value = (value + 512)/1024;
641         n++;
642     }
643
644     if ((0 != value) && (0 == value/1024)) {
645         value = (value + 512)/1024;
646         n++;
647     }
648
649     (void) sprintf(temp_buffer, "%ldtc", value, suffix[n]);
650
651     if (strlen(temp_buffer) > size)
652         (void) strcpy(buffer, "");
653     else
654         (void) strcpy(buffer, temp_buffer);
655
656     return (buffer);
657 }
658
659
660 instances_per_lane(instances, faults)
661     long instances[];
662     long faults[];
663
664 {
665     long i, j, k;
666     long number_of_devices, *list_of_device_numbers;
667     long inst, flts, dabs, device_number, locs, *lanes, *slots;
668     char **list_of_device_names;
669
670     for (i = 0; i < 4; i++) {
671         instances[i] = 0;
672         faults[i] = 0;
673     }
674
675     LM_CHECK(lm_inquire_device_list(&number_of_devices, &list_of_device_numbers, &list_of_device_names));
676
677     for (i = 0; i < number_of_devices; i++) {
678         device_number = list_of_device_numbers[i];
679         LM_CHECK(lm_inquire_device(device_number, LM_NUMBER_OF_ACTIVE_INSTANCES, &inst));
680         LM_CHECK(lm_inquire_device(device_number, LM_NUMBER_OF_ACTIVE_FAULTS, &flts));
681         LM_CHECK(lm_inquire_device(device_number, LM_NUMBER_OF_DABS, &dabs));
682         for (j = 0; j < dabs; j++) {
683             LM_CHECK(lm_inquire_dab_location(device_number, j, &locs, &lanes, &slots));
684             for (k = 0; k < locs; k++) {
685                 instances[lanes[k]] += inst;
686                 faults[lanes[k]] += flts;
687             }
688         }
689     }
690
691     return (SUCCESS);
692 }
693
694
695 display_logic_models(modeler_name)
696     char modeler_name[];
697
698 {
699     char locations[80], *c;
700     long number_of_devices, *list_of_device_numbers;
701     char **list_of_device_names;
702     char *ret_string;
703     printf("device_array");
704     long dev_status;
705     long active_i;
706     long active_f;
707     char dab_type[80];
708     char dab_rev[80];
709     char sfx[80];
710     long locs, *lanes, *slots;
711     long i, j;
712     long device_number;
713
714     /* for each modeler... */
715
716     (void) printf("\n");
717     (void) printf("\t\t\t\t\t LM-1000 Logic Models \n", modeler_name);
718     (void) printf("\n");
719     (void) printf("\t\t\t\t\t Logic Model", "Lanes", "Device", " Simulator", "Adapter", "Dev.", "Rev.");
720     (void) printf("\t\t\t\t\t blank, blank, blank, blank, blank, blank, blank, blank");

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89

PAGE #

TIME 1:20:34 pm

7/20

```

LINE # SOURCE TEXT
721 (void) printf("\n");
722
723 LM_CHECK(lm_inquire_device_list(number_of_devices,
724 list_of_device_numbers, list_of_device_names));
725
726 if (number_of_devices == 0) {
727 (void) printf("No devices installed\n");
728 return SUCCESS;
729 }
730 device_array = (printline *) malloc ((unsigned int) number_of_devices * sizeof(printline));
731
732 for (i = 0; i < number_of_devices; i++) {
733 device_number = list_of_device_numbers[i];
734 LM_CHECK_TO(log_mod_cleanup, lm_inquire_device(device_number,
735 LM_DEVICE_STATUS, &dev_status));
736 LM_CHECK_TO(log_mod_cleanup, lm_inquire_device(device_number,
737 LM_NUMBER_OF_ACTIVE_INSTANCES, &active_i));
738 LM_CHECK_TO(log_mod_cleanup, lm_inquire_device(device_number,
739 LM_NUMBER_OF_ACTIVE_FAULTS, &active_f));
740 LM_CHECK_TO(log_mod_cleanup, lm_inquire_dab(device_number,
741 (long) 0, LM_DAB_TYPE, &ret_string));
742 (void) strcpy(dab_type, ret_string);
743 LM_CHECK_TO(log_mod_cleanup, lm_inquire_dab(device_number,
744 (long) 0, LM_DAB_REVISION, &ret_string));
745 (void) strcpy(dab_rev, ret_string);
746 LM_CHECK_TO(log_mod_cleanup, lm_inquire_dab(device_number,
747 (long) 0, LM_DAB_MAKER, &ret_string));
748 (void) strcpy(mfr, ret_string);
749 LM_CHECK_TO(log_mod_cleanup,
750 lm_inquire_dab_location(device_number,
751 (long) 0, &locs, &slots));
752
753 c = locations;
754 for (j = 0; j < locs; j++) {
755 *c++ = 'L' + lanes[j];
756 *c++ = 'S' + slots[j];
757 if (j < locs - 1)
758 *c++ = ',';
759 }
760 *c = '\0';
761 (void) sprintf((char *) (device_array + i), fmt5, "list_of_device_names++",
762 locations,
763 (dev_status == LM_PRIVATE) ? "PRIVATE" : "PUBLIC",
764 active_i, active_f, dab_type, mfr, dab_rev);
765
766 #ifdef VMS
767 qsort((char *) device_array, (int) number_of_devices, sizeof(printline), str_cmp);
768 #else
769 vms_qsort((char *) device_array, (int) number_of_devices, sizeof(printline), vms_str_cmp);
770 #endif
771
772 for (i = 0; i < number_of_devices; i++) {
773 (void) printf("%s", device_array + i);
774 }
775
776 log_mod_cleanup:
777 (void) printf("\n");
778 free((char *) device_array);
779
780 return (SUCCESS);
781
782
783
784
785 #define LANE_OFFSET 0x80000000
786 #define PAM_SIZE 0x00000100
787 #define PEL_SIZE 0x00001000
788 #define PEL_OFFSET 0x00000000
789
790 display_version(modeler_name)
791 char *modeler_name;
792
793 {
794 (void) printf("\n");
795 (void) printf("LM-1000 Versions\n", modeler_name);
796 LM_CHECK(lm_select_modeler(modeler_name));
797
798 (void) printf("Module Board ID Revision\n");
799 (void) printf("\n");
800
801 if (FAILURE == display_cpu_revision(modeler_name))
802 return (FAILURE);
803 if (FAILURE == display_tmg_revision(modeler_name))
804 return (FAILURE);
805
806 (void) printf("\n");
807 if (FAILURE == display_pac_revisions(modeler_name))
808 return (FAILURE);
809
810 (void) printf("\n");
811 if (FAILURE == display_pam_revisions(modeler_name))
812 return (FAILURE);
813
814 (void) printf("\n");
815 if (FAILURE == display_pel_revisions(modeler_name))
816 return (FAILURE);
817
818 (void) printf("\n");
819 if (FAILURE == display_pel_revisions(modeler_name))
820 return (FAILURE);
821
822 (void) printf("\n");
823 return (SUCCESS);
824
825
826
827
828 display_cpu_revision(modeler_name)
829 char *modeler_name;
830
831 {
832 ID_PROM_CPU cpu_id_struct;
833
834 read_every_forth_byte(modeler_name, (u_long) CPU_ID_PROM,
835 (char *) &cpu_id_struct, (u_long) sizeof(cpu_id_struct));
836
837 if (ID_CHECKSUM_GOOD == id_checksum((u_char *) &cpu_id_struct)) {
838 (void) printf("CPU ID: %4d %4s\n",
839 cpu_id_struct.generic.board_type,
840 create_revision(cpu_id_struct.generic.revision, cpu_id_struct.generic.eco_level));
841 }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89

PAGE #

TIME 1:20:34 pm

8/21

```

LINE # SOURCE TEXT
841 } else {
842     (void) printf(" CPU
843     }
844     return (SUCCESS);
845 }
846
847
848
849
850 display_tmz_revisions(modular_name)
851 char *modular_name;
852
853 {
854     ID_FROM_TMZ tmz_id_struct;
855     read_every_forth_byte(modular_name, (u_long) TMZ_ID_FROM,
856     (char *) &tmz_id_struct, (u_long) sizeof(tmz_id_struct));
857
858     if (ID_CHECKSUM_GOOD == id_checksum((u_char *) &tmz_id_struct)) {
859         (void) printf(" Timing Generator
860         (void) printf(" Timing Generator
861         tmz_id_struct.generic.board_type,
862         create_revision (tmz_id_struct.generic.revision, tmz_id_struct.generic.eco_level));
863     } else {
864         (void) printf(" Timing Generator
865         }
866     }
867     return (SUCCESS);
868 }
869
870
871 display_pac_revisions(modular_name)
872 char *modular_name;
873
874 {
875     long lane;
876     long number_of_lanes;
877     long has_a_pac;
878     ID_FROM_PAC pac_id_struct;
879
880     LM_CHECK(lm_inquire_modelar(LM_NUMBER_OF_LANES, &number_of_lanes));
881     for (lane = 0; lane < number_of_lanes; lane++) {
882         LM_CHECK(lm_inquire_lane(lane, LM_IS_LANE_USABLE, &has_a_pac));
883         if (LM_TRUE == has_a_pac) {
884             read_every_forth_byte(modular_name, (u_long) LANE_A_PAC_ID_FROM + LANE_SIZE * lane,
885             (char *) &pac_id_struct, (u_long) sizeof(pac_id_struct));
886             if (ID_CHECKSUM_GOOD == id_checksum((u_char *) &pac_id_struct)) {
887                 (void) printf(" Pattern Controller, Lane %c
888                 lane + 'A', pac_id_struct.generic.board_type,
889                 create_revision (pac_id_struct.generic.revision, pac_id_struct.generic.eco_level));
890             } else {
891                 (void) printf(" Pattern Controller, Lane %c
892                 lane + 'A';
893             }
894         }
895     }
896     return (SUCCESS);
897 }
898
899
900 display_pam_revisions(modular_name)
901 char *modular_name;
902
903 {
904     long lane;
905     long number_of_lanes;
906     long pams, pam, has_a_pac;
907     long pam_size;
908     char b[10];
909     ID_FROM_PAM pam_id_struct;
910
911     LM_CHECK(lm_inquire_modelar(LM_NUMBER_OF_LANES, &number_of_lanes));
912     for (lane = 0; lane < number_of_lanes; lane++) {
913         LM_CHECK(lm_inquire_lane(lane, LM_IS_LANE_USABLE, &has_a_pac));
914         if (LM_TRUE == has_a_pac) {
915             LM_CHECK(lm_inquire_lane(lane, LM_NUMBER_OF_PAMS, &pams));
916             for (pam = 0; pam < pams; pam++) {
917                 LM_CHECK(lm_inquire_pam(lane, pam, LM_PAM_MEMORY_SIZE, &pam_size));
918                 read_every_forth_byte(modular_name, (u_long) LANE_A_PAM_ID_FROM + LANE_SIZE * lane + PAM_SIZE
919                 * pam,
920                 (char *) &pam_id_struct, (u_long) sizeof(pam_id_struct));
921                 if (ID_CHECKSUM_GOOD == id_checksum((u_char *) &pam_id_struct)) {
922                     (void) printf(" %s Fast Pattern Memory, Lane %c strap id
923                     string_it(pam_size, b, 4), lane + 'A', pam, pam_id_struct.generic.board_type,
924                     create_revision (pam_id_struct.generic.revision, pam_id_struct.generic.eco_level));
925                 } else {
926                     (void) printf(" %s Fast Pattern Memory, Lane %c strap id
927                     string_it(pam_size, b, 4), lane + 'A', pam;
928                     ID FROM READ ERROR --- \n";
929                 }
930             }
931         }
932     }
933     return (SUCCESS);
934 }
935
936
937 display_pel_revisions(modular_name)
938 char *modular_name;
939
940 {
941     long lane;
942     long slot;
943     long number_of_slots;
944     long number_of_lanes;
945     long has_a_pac;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
9/22

```

LINE # SOURCE TEXT
961 long has_a_pel;
962 ID_FROM_PEL pel_id_struct;
963
964 LM_CHECK(lm_inquire_modeler(LM_NUMBER_OF_LANES, &number_of_lanes));
965 LM_CHECK(lm_inquire_modeler(LM_NUMBER_OF_SLOTS_PER_LANE, &number_of_slots));
966
967 for (lane = 0; lane < number_of_lanes; lane++) {
968     LM_CHECK(lm_inquire_lane(lane, LM_IS_LANE_USABLE, &has_a_pel));
969     if (LM_TRUE == has_a_pel) {
970         for (slot = 0; slot < number_of_slots; slot++) {
971             LM_CHECK(lm_inquire_pel(lane, slot, LM_DOES_PEL_EXIST, &has_a_pel));
972             if (LM_TRUE == has_a_pel) {
973                 read_every_forth_byte(modeler_name, (u_long) PEL_ID_FROM + LANE_OFFSET + PEL_OFFSET
974                     LANE_SIZE * lane + PEL_SIZE * slot.
975                     (char *) &pel_id_struct, (u_long) sizeof(pel_id_struct));
976                 if (ID_CHECKSUM_GOOD == id_checksum((u_char *) &pel_id_struct)) {
977                     (void) printf(" Pin Electronics Module, Lane %c Slot %d      %4d %4s\n",
978                         lane + 'A', slot, pel_id_struct.generic.board_type,
979                         create_revision(pel_id_struct.generic.revision, pel_id_struct.generic.eco_level));
980                 } else {
981                     (void) printf(" Pin Electronics Module, Lane %c Slot %d      ID FROM READ ERROR --- \n",
982                         lane + 'A', slot);
983                 }
984             }
985         }
986     }
987 }
988
989 return (SUCCESS);
990
991 static char *
992 create_revision(revision, eco_level)
993     unsigned char revision;
994     unsigned char eco_level;
995 {
996     static char buffer[10];
997     char eco_str[10];
998     if ((eco_level >= ID_MIN_ECO) && (eco_level <= ID_MAX_ECO))
999         (void) sprintf(eco_str, "%d", eco_level);
1000     else
1001         (void) sprintf(eco_str, "ec", eco_level);
1002     (void) sprintf(buffer, "%04s", revision, eco_str);
1003     return(buffer);
1004 }
1005
1006 id_checksum(address)
1007     u_char *address;
1008 {
1009     register long checksum;
1010     register u_long byte_count;
1011     checksum = ID_CHECKSUM_INIT;
1012     for (byte_count = 0; byte_count < ID_NUM_BYTES; byte_count++) {
1013         checksum = (checksum << 1) + ((checksum & 0x80) >> 7);
1014         checksum += *(address + byte_count);
1015         checksum &= 0xFF;
1016     }
1017     return (checksum);
1018 }
1019
1020 #ifdef VMS
1021 extern int SORSM_STABLE, SORSM_NOSIGNAL;
1022
1023 vms_qsort (base, elements, size, comparison)
1024     char *base;
1025     int elements;
1026     int size;
1027     int (*comparison)();
1028 {
1029     long sort_status;
1030     int i;
1031     long context;
1032     short length;
1033     char *current_record;
1034     struct dscibdescriptor data_descriptor;
1035     short short_size;
1036     long flags;
1037     long file_blocks;
1038     char *sum_files;
1039     sum_files = 2;
1040     file_blocks = 100;
1041     flags = 0;
1042     short_size = size;
1043     current_record = base;
1044     context = 0;
1045     length = size - 1;
1046     flags = SORSM_STABLE + SORSM_NOSIGNAL;
1047     data_descriptor.dscib_length = size;
1048     data_descriptor.dscib_dtype = DSCIB_DTYPE_2;
1049     data_descriptor.dscib_class = DSCIB_CLASS_5;
1050     sort_status = SORSBEGIN_SORT(0, &short_size, &flags,
1051         &file_blocks, comparison, 0, 0, &sum_files, &context);
1052     for (i=0; i<elements; i++) {
1053         data_descriptor.dscib_pointer = current_record;
1054         sort_status = SORSELE_RELEASE_REC (&data_descriptor, &context);
1055         current_record += size;
1056     }
1057 }

```

000045

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/display.c

DATE 5/23/89
TIME 1:20:34 pm

PAGE #
10/23

LINE #	SOURCE TEXT
1081	
1082	sort_status = SORSSORT_MERGE (&context);
1083	
1084	current_record = base;
1085	
1086	for (i=0; i<elements; i++) {
1087	data_descriptor.descsa_pointer = current_record;
1088	sort_status = sorreturn_rec (&data_descriptor, &length, &context);
1089	current_record += size;
1090	}
1091	
1092	sort_status = SORSEND_SORT (&context);
1093	
1094	}
1095	
1096	
1097	vms_str_cmp (s1, s2, l1, l2, context) /* must be called by vms_qsort() */
1098	char *s1, *s2;
1099	int l1, l2, context;
1100	
1101	{
1102	if (str_cmp (s1, s2) < 0) return (-1);
1103	if (str_cmp (s1, s2) > 0) return (1);
1104	return(0);
1105	
1106	}
1107	
1108	#endif
1109	

Copyright 1989

Logic Modeling Systems

HEADER FILE

lm/help.h

DATE

5/23/89

PAGE #

TIME

1:20:35 pm

1/24

```

LINE #
1  // SCCS_ID: help.h rev 1.1 4/24/89 at 07:51:56
2  // help file for the logic based utilities
3
4  /*
5   * All single quotes have been removed from this file.
6   * The Apollo STS compiler/preprocessor has a bug in its
7   * handling of single quotes in some double-quoted strings.
8   * Their removal is really only in two places: where
9   * ownership is represented by an "a". ld_help and in help
10  * are the two cases of this. I guess that's life in the
11  * portable code world.
12  */
13
14  static char ld_help[] =
15  {
16      "This function labels a Device Adapter with the device name \n\
17      to be used for a custom Logic Model. The device name must \n\
18      exactly match the \"device_name\" used in the Logic Model \n\
19      Shell Software. The function will prompt for the following \n\
20      information: \n\
21      \n\
22      Modeler Name - the name of the modeler where \n\
23      the Device Adapter is installed. \n\
24      \n\
25      Lane (A - D) - the lane where the Device Adapter \n\
26      is installed. \n\
27      \n\
28      Slot (0 - 7) - the slot where the Device Adapter \n\
29      is installed. \n\
30      \n\
31      Device Name - the name of the model device mounted on \n\
32      the Device Adapter. \n\
33      \n\
34      Model Developer - the name of the company or person \n\
35      that developed the Logic Model. \n\
36      \n\
37      Model Revision - the revision number of the Logic Model. \n\
38      \n\
39      ",
40  };
41
42  static char cs_help[] =
43  {
44      "This function performs a system check on the specified \n\
45      Shell Software, and returns a list of any errors encountered \n\
46      while attempting to parse the Shell Software. It will \n\
47      prompt for the following information: \n\
48      \n\
49      Model File (xxx.MDL) - the name of the file which \n\
50      contains the Shell Software to be checked. \n\
51      \n\
52      ",
53  };
54
55  static char ct_help[] =
56  {
57      "This function creates a timing (.TME) file, which is suitable for \n\
58      usage as Shell Software, from timing measurement output file(s). \n\
59      It will prompt for the following information: \n\
60      \n\
61      TIM File(s) - a list of one or more timing measurement \n\
62      output files, separated by commas or spaces. \n\
63      \n\
64      TME File - the name of the resulting Shell Software \n\
65      timing file. \n\
66      \n\
67      Combine Bus Delays - yes or no. Combining bus delays \n\
68      combines the measured delays for all pins \n\
69      names which have the same prefix \n\
70      (e.g. data0 through data31). \n\
71      \n\
72      ",
73  };
74
75  static char pv_help[] =
76  {
77      "This function presents test vectors to a specified device \n\
78      from a test vector file. Any discrepancies between the measured \n\
79      output and the expected results (specified in the vector file) \n\
80      are reported. The function will prompt for the following \n\
81      information: \n\
82      \n\
83      Model File (xxx.MDL) - the name of the file which \n\
84      contains the Shell Software for the \n\
85      device to be tested. \n\
86      \n\
87      Vector File (xxx.TST) - the name of the file which \n\
88      contains the test vectors. \n\
89      \n\
90      Output File (xxx.OUT) - the name of the file in which \n\
91      to log any discrepancies detected while \n\
92      playing the vectors to the device. If no \n\
93      file name is specified, the discrepancies \n\
94      are displayed on the screen. \n\
95      \n\
96      ",
97  };
98
99  static char mt_help[] =
100  {
101      "This function measures the actual propagation delays on a \n\
102      Logic Model device. It presents test vectors to the specified \n\
103      device from a test vector file, and measures the resulting output \n\
104      pin delays. These delays are stored in a timing measurement output \n\
105      file. The function will prompt for the following information: \n\
106      \n\
107      Model File (xxx.MDL) - the name of the file which \n\
108      contains the Shell Software for the \n\
109      device to be measured. \n\
110      \n\
111      Vector File (xxx.TST) - the name of the file which \n\
112      contains the device test vectors. \n\
113      \n\
114      Timing File (xxx.TIM) - the name of the file in which \n\
115      to log the measured delays. \n\
116      \n\
117      Output File (xxx.OUT) - the name of the file in which \n\
118      to log any discrepancies detected while \n\
119      playing vectors to the device. If no \n\
120      file name is specified, the discrepancies \n\
121      are displayed on the screen. \n\
122      \n\
123      ",
124  };
125
126  static char fm_help[] =
127  {
128      "This function is used to locate modeler files. The \n\
129      \n\
130      ",
131  };

```

Copyright 1989 Logic Modeling Systems		HEADER FILE lm/help.h	DATE 5/23/89	PAGE # 2/25
LINE #	HEADER TEXT			
121	function searches the LM_BIN and LM_LIB paths for a given \n\			
122	filename. If the file is found, the full pathname of the file \n\			
123	is printed. If not found, an error message is displayed. The \n\			
124	function will prompt for the following information: \n\			
125	\n\			
126	File Name - the name of the file to be located. \n\			
127	NOTE: Wildcard characters are not allowed \n\			
128	as part of the filename. \n\			
129	,			
130	static char in_help[] =			
131	132			
133	134 "This function is used to install a modeler. It stores the \n\			
135	modeler boot information in the non-volatile RAM in the modeler. \n\			
136	The function will prompt for the following information: \n\			
137	\n\			
138	Modeler Name - the name of the modeler to be \n\			
139	installed. \n\			
140	\n\			
141	Autoboot - yes or no. If set to "yes" the modeler \n\			
142	will initiate the boot process, otherwise \n\			
143	the modeler must be manually booted using \n\			
144	Boot Modeler. \n\			
145	\n\			
146	Boot Host Name - the name of the host computer to send \n\			
147	network requests. The specified host must \n\			
148	be running the lmdemon program. \n\			
149	,			
150	static char bt_help[] =			
151	152			
153	154 "This function manually boots the modeler by downloading the \n\			
155	Core Modeler Code files to the modeler. It will prompt for the \n\			
156	following information: \n\			
157	\n\			
158	Modeler Name - the name of the modeler to be \n\			
159	booted. \n\			
160	,			
161	static char sh_help[] =			
162	163			
164	165 "This function performs an orderly shutdown of the specified \n\			
166	modeler. The modeler will remain down until it is manually \n\			
167	rebooted. The function will prompt for the following information: \n\			
168	\n\			
169	Modeler Name - the name of the modeler to be \n\			
170	shut down. \n\			
171	\n\			
172	Wait - if there are other users on the \n\			
173	system, enter "yes" to wait for all \n\			
174	current simulation sessions to complete. \n\			
175	\n\			
176	Minutes - number of minutes to wait before \n\			
177	shutting down the modeler. \n\			
178	,			
179	static char rh_help[] =			
180	181			
182	183 "This function reboots the specified modeler. It will prompt \n\			
184	for the following information: \n\			
185	\n\			
186	Modeler Name - the name of the modeler to be \n\			
187	rebooted. \n\			
188	\n\			
189	Wait - if there are other users on the \n\			
190	system, enter "yes" to wait for all \n\			
191	current simulation sessions to complete. \n\			
192	\n\			
193	Minutes - number of minutes to wait before \n\			
194	rebooting the modeler. \n\			
195	,			
196	static char sb_help[] =			
197	198			
199	199 "This function sets the baud rate for either the console \n\			
200	port or the modem port. It displays a menu which allows \n\			
201	the selection of the port and baud rate. \n\			
202	,			
203	static char st_help[] =			
204	205			
206	206 "This function sets the inactive process timeout period on \n\			
207	the modeler. If a process on the modeler performs no activity \n\			
208	for this period of time, the modeler will attempt to determine \n\			
209	if the corresponding host process is still running. If the host \n\			
210	process is no longer running then the process on the modeler \n\			
211	is aborted. The function will prompt for the following: \n\			
212	\n\			
213	Modeler Name - the name of the modeler on which to set \n\			
214	the timeout. \n\			
215	\n\			
216	Minutes - the number of minutes that a process \n\			
217	may be inactive before it is cleared. \n\			
218	,			
219	static char su_help[] =			
220	221			
222	222 "This function aborts the specified modeler process. It will \n\			
223	prompt for the following information: \n\			
224	\n\			
225	Modeler Name - the name of the modeler on which to \n\			
226	abort the process. \n\			
227	\n\			
228	User Number - the user number of the process to be \n\			
229	deleted. This number can be determined \n\			
230	using "Show Users". \n\			
231	,			
232	static char pw_help[] =			
233	234			
235	235 "This function sets/modifies the modeler password which \n\			
236	protects utility functions which may affect other modeler users. \n\			
237	It will prompt for the following information: \n\			
238	\n\			
239	Modeler Name - the name of the modeler on which to \n\			
240	set/modify the password. \n\			
241	\n\			
242	Enter OLD Modeler Password - the old modeler password. \n\			

Copyright 1989
Logic Modeling Systems

HEADER FILE
lm/help.h

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
3/26

LINE #	HEADER TEXT
241	The password will not be echoed to the \n\
242	screen. If there was no password, enter \n\
243	a carriage return.\n\
244	\n\
245	Enter NEW Modeler Password - the new modeler password. \n\
246	To disable password protection, enter \n\
247	a carriage return.\n\
248	\n\
249	Repeat NEW Modeler Password - repeat the new modeler \n\
250	password to verify that the password was \n\
251	entered correctly.\n\
252	."
253	
254	
255	static char cl_help[] =
256	
257	"This selection presents a menu of functions used to create \n\
258	Logic Models. These functions include Device Adapter labeling \n\
259	and Shell Software syntax checking. \n\
260	."
261	
262	static char vl_help[] =
263	
264	"This selection presents a menu of functions used to verify \n\
265	the integrity of existing Logic Models. These functions include \n\
266	playing test vectors and measuring device timing. \n\
267	."
268	
269	static char mw_help[] =
270	
271	"This selection presents a menu of functions used to perform \n\
272	modeler management. These functions include booting, configuration, \n\
273	and installation of the modeler. \n\
274	."
275	
276	static char so_help[] =
277	
278	"This selection presents a menu of functions used to display \n\
279	various modeler configuration and usage information. \n\
280	."
281	
282	static char rd_help[] =
283	
284	"This function boots and executes the modeler diagnostics. The \n\
285	diagnostics exercise the hardware to assist in isolating \n\
286	hardware related problems. The function will prompt for \n\
287	the following information: \n\
288	\n\
289	Modeler Name - the name of the modeler to boot with the \n\
290	diagnostics. \n\
291	."
292	
293	static char su_help[] =
294	
295	"This function displays the configuration of the modelers \n\
296	available on the network. It will prompt for the following \n\
297	information: \n\
298	\n\
299	Modeler Name - the name of the modeler to be queried. \n\
300	By default, all modelers are displayed.\n\
301	."
302	
303	static char al_help [] =
304	
305	"This function displays the Logic Models currently installed in \n\
306	a modeler. It will prompt for the following information: \n\
307	\n\
308	Modeler Name - the name of the modeler to be queried. \n\
309	By default, all modelers are displayed.\n\
310	."
311	
312	static char su_help[] =
313	
314	"This function displays information about the modeler users. \n\
315	It will prompt for the following information: \n\
316	\n\
317	Modeler Name - the name of the modeler to be queried. \n\
318	By default, all modelers are displayed.\n\
319	."
320	
321	static char sv_help[] =
322	
323	"This function displays revision information about the boards \n\
324	installed in a modeler. It will prompt for the following \n\
325	information: \n\
326	\n\
327	Modeler Name - the name of the modeler to be queried. \n\
328	By default, all modelers are displayed.\n\
329	."

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89

PAGE #

TIME 1:20:35 pm

1/27

LINE # SOURCE TEXT

```

1  /* SCCS ID: hostdiag.c rev 3.1, 4/24/89, at 07:52:01 */
2
3  #include <stdio.h>
4  #include <sys/ioctl.h>
5  #include <sys/jmp.h>
6  #include <signal.h>
7  #include <time.h>
8  #include <ctype.h>
9  #ifdef SYS5
10 #include <unistd.h>
11 #include <fcntl.h>
12 #endif /*SYS5*/
13 #include "common.h"
14 #include "mod_err.h"
15 #include "warn.h"
16 #include "message.h"
17 #include "network.h"
18 #include "lm_diags.h"
19 #include "lm_err.h"
20 #include "lm_test.h"
21
22 extern int goodbye();
23 extern int lm_suspend();
24 extern void clean_input();
25 extern void get_line();
26 extern char *calloc();
27
28 extern u_long lm_network_timeout;
29 extern u_long lm_max_timeout_retry_attempts;
30
31 static lm_save_timeout;
32 static lm_save_retrys;
33 static lm_boot_mode;
34
35 CONNECTION *conn;
36 char selection_buffer[64];
37
38 #define DIAG_MAX_TRIES 30
39
40 #ifdef VMS
41 #define CLEANUP \
42     jmpbuf = save_jmpbuf; \
43     \
44     (void) signal(SIGINT, save_intr_fn); \
45     (void) signal(SIGUP, save_hup_fn); \
46     (void) signal(SIGTERM, save_term_fn); \
47     \
48     lm_network_timeout = lm_save_timeout; \
49     lm_max_timeout_retry_attempts = lm_save_retrys;
50 #else
51 #define CLEANUP \
52     jmpbuf = save_jmpbuf; \
53     \
54     (void) signal(SIGINT, save_intr_fn); \
55     (void) signal(SIGUP, save_hup_fn); \
56     (void) signal(SIGTERM, save_term_fn); \
57     (void) signal(SIGTSTP, save_stp_fn); \
58     \
59     lm_network_timeout = lm_save_timeout; \
60     lm_max_timeout_retry_attempts = lm_save_retrys; \
61     \
62     if (! power_up_mode) cook();
63 #endif
64
65 #define DIAG_CHECK(fn) \
66     { \
67         int status; \
68         \
69         status = fn; \
70         if (status != SUCCESS) return(FAILURE); \
71     }
72
73
74
75
76 lm_diagnostics(modular_name, error_count, cmd_list, power_up_mode)
77 char *modular_name;
78 char *cmd_list;
79 int *error_count;
80 int power_up_mode;
81
82 {
83     long status;
84     int handle_intr();
85 #ifdef SUSV3_5
86     int (*save_intr_fn)();
87     int (*save_hup_fn)();
88     int (*save_term_fn)();
89     int (*save_stp_fn)();
90 #else
91     void (*save_intr_fn)();
92     void (*save_hup_fn)();
93     void (*save_term_fn)();
94     void (*save_stp_fn)();
95 #endif
96     jmp_buf jmpbuf;
97     jmp_buf jmpbuf;
98     jmp_buf *save_jmpbuf = jmpbuf;
99     int try;
100     char *s;
101     char host_name[32], *user_name, User_name[32];
102
103     lm_boot_mode = power_up_mode;
104     if (0 != gethostname(host_name, sizeof(host_name))) {
105         strcpy(host_name, "unknown host");
106     }
107
108     user_name = (char *) getenv ((char *) "USER");
109     if (user_name)
110         strcpy (User_name, user_name);
111     else
112         strcpy (User_name, "unknown user");
113
114     conn = 0;
115     if (error_count) *error_count = 0;
116
117     if (!setjmp(jmpbuf)) {
118         jmpbuf = (jmp_buf *) jmpbuf;
119         /* save interrupts and timeouts */
120

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
2/28

```

121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240

SOURCE TEXT

save_intr_fa = signal (SIGINT, handle_intr);
save_hup_fa = signal (SIGHUP, good_bye);
save_term_fa = signal (SIGTERM, good_bye);

#ifdef VMS
save_stp_fa = signal (SIGSTP, lm_suspend);
#endif

lm_save_timeout = lm_network_timeout;
lm_save_retrys = lm_max_timeout_retry_attempts;

if (! power_up_mode) raw ();

conn = (CONNECTION *) calloc ((unsigned) 1, sizeof(CONNECTION));
conn->name = modeler_name;

(void) printf("Requesting connection...\n");

for (try = 0; try < DIAG_MAX_TRIES; ++try) {
    status = lm_check((int) lm_init_connection_for_client(conn));
    if (status != SUCCESS) {
        CLEANUP;
        free((char *) conn);
        return(FAILURE);
    }

    lm_network_timeout = 1000; /* 1 second. Lets be responsive. */
    lm_max_timeout_retry_attempts = 1;
    LM_CHK_PUT_LONG(conn, LM_DIAG_CIMSE);
    LM_CHK_PUT_LONG(conn, 4 + strlen(host_name) + strlen(user_name) + 2);

    s = user_name;
    while (*s) {
        LM_CHK_PUT_CHAR(conn, *s);
        s++;
    }
    LM_CHK_PUT_CHAR(conn, '\0');

    s = host_name;
    while (*s) {
        LM_CHK_PUT_CHAR(conn, *s);
        s++;
    }
    LM_CHK_PUT_CHAR(conn, '\0');

    if (lm_send_request_get_reply(conn) == SUCCESS) {
        /* connection established */
        (void) printf("Connection established.\n");
        (void) sprintf(selection_buffer,
            "%s selection: ", modeler_name);

        lm_network_timeout = 30000; /* 30 seconds */
        lm_max_timeout_retry_attempts = 2000;
        while (process_request(conn, error_count,
            &cmd_list, power_up_mode) == SUCCESS)
            break;

    } else {
        lm_close_connection_for_client(conn);
    }

} else {
    char state;
    int i;

    /* try twice to give lance a chance */
    /* (this might be right after boot up) */
    for (i = 0; i < 2; ++i) {
        if (lm_is_modeler_alive(modeler_name, &state)
            == LM_SUCCESS) {
            /* only kill if running diagnostics */
            if (state == RUNNING_DIAGS) {
                if (lm_remote_reset(modeler_name,
                    &state) == LM_SUCCESS)
                    break;
            } else break;
        }
    }

    if (conn) {
        lm_close_connection_for_client(conn);
        free((char *) conn);
    }
    CLEANUP;
}

#ifdef VMS
save_intr_fa(SIGINT);
#endif

if (try == DIAG_MAX_TRIES) {
    (void) printf("Connection failed.\n");
} else {
    lm_close_connection_for_client(conn);
}

CLEANUP;

if (conn)
    free((char *) conn);
if (try == DIAG_MAX_TRIES) return (FAILURE);
return(SUCCESS);

#ifdef SYS5
static unsigned char c_EOF, c_EOL;
raw ()
{
    struct termio term;
    if (isatty (0))
        if (ioctl (0, TCGETA, &term) == -1)
            perror ("raw stdin");
    term.c_iflag &= ~ICANON;
    term.c_lflag &= ~ECHO;
}

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
3/29

```

LINE #
241 c_EOF = term . c_cc [ 4 ] ;
242 c_EOL = term . c_cc [ 5 ] ;
243 term . c_cc [ 4 ] = 1 ;
244 term . c_cc [ 5 ] = 0 ;
245 if ( ioctl ( 0 , TCSETA , & term ) == -1 )
246     perror ( "raw stdin 2" ) ;
247 } /* raw */
248
249 cook ( )
250 {
251     struct termio term ;
252     if ( isatty ( 0 ) )
253     {
254         if ( ioctl ( 0 , TCSETA , & term ) == -1 )
255             perror ( "cook stdin 1" ) ;
256         term . c_iflag |= ICANON ;
257         term . c_lflag |= ECHO ;
258         term . c_cc [ 4 ] = c_EOF ;
259         term . c_cc [ 5 ] = c_EOL ;
260         if ( ioctl ( 0 , TCSETA , & term ) == -1 )
261             perror ( "cook stdin 2" ) ;
262     } /* cook */
263 # else /* SYS5 */
264     raw ( )
265 {
266     struct sgttyb tty ;
267 #ifdef VMS
268     if ( isatty(0) ) {
269         if ( ioctl(0, TIOCGETP, &tty) == -1 ) {
270             perror("raw stdin 1");
271         }
272         tty.sg_flags |= (CBREAK);
273         tty.sg_flags |= (ECHO);
274         if ( ioctl(0, TIOCSETP, &tty) == -1 ) {
275             perror("raw stdin 2");
276         }
277     }
278 #endif
279 }
280 #endif
281
282 cook ( )
283 {
284     struct sgttyb tty ;
285 #ifdef VMS
286     if ( isatty(0) ) {
287         if ( ioctl(0, TIOCGETP, &tty) == -1 ) {
288             perror("cook stdin 1");
289         }
290         tty.sg_flags |= (CBREAK);
291         tty.sg_flags |= (ECHO);
292         if ( ioctl(0, TIOCSETP, &tty) == -1 ) {
293             perror("cook stdin 2");
294         }
295     }
296 #endif
297 #endif /* SYS5 */
298
299 int
300 lm_suspend ( )
301 {
302 #ifdef VMS
303     cook ( ) ;
304     (void) signal(SIGTSTP, SIG_DFL);
305     if ( sigsetmask(0) == -1 ) {
306         perror("sigsetmask");
307     }
308     (void) kill (getpid(), SIGTSTP);
309     (void) signal(SIGTSTP, lm_suspend);
310     raw ( ) ;
311     return(SUCCESS);
312 #endif
313 }
314
315 static int
316 good_bye ( )
317 {
318     int pgrp ;
319 #ifdef VMS
320 #ifdef SYS5
321     if ( setpgp ( ) == -1 )
322         perror ( "setpgp" ) ;
323 # else /* SYS5 */
324     (void) ioctl(0, TIOCGPGRP, &pgrp);
325     if ( setpgp(0, pgrp) == -1 ) {
326         /* make sure we can cook */
327         perror("setpgp");
328     }
329 #endif /* SYS5 */
330 #endif /* VMS */
331     cook ( ) ;
332     (void) signal(SIGHUP, SIG_DFL);
333     (void) signal(SIGINT, SIG_DFL);
334     (void) signal(SIGTERM, SIG_DFL);
335 #ifdef VMS
336     (void) signal(SIGTSTP, SIG_DFL);
337     if ( sigsetmask(0) == -1 ) {

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

lm/hostdiag.c

DATE

5/23/89

PAGE #

TIME

1:20:35 pm

4/30

```

LINE # SOURCE TEXT
361 perror("sigsetmask");
362
363 #endif /*VMS*/
364
365 lm_chk_put_long(conn, lm_diag_1
366 lm_chk_put_long(conn, 4);
367 (void)lm_chk((int)lm_send_request_get_reply(conn));
368
369 u_exit(); /* u_exit never returns, so good bye */
370
371
372
373 process_request(conn, error_count, cmd_list, power_up_mode)
374 {
375     CONNECTION
376     conn;
377     int
378     error_count;
379     char
380     cmd_list;
381     int
382     power_up_mode;
383
384     int i, size, cmd;
385     char line(max_str);
386     char str(max_str);
387     char c;
388
389     lm_chk_put_long(conn, lm_diag_ginck);
390     lm_chk_put_long(conn, 4);
391     DIAG_CHECK((int)lm_send_request_get_reply(conn));
392
393     while (1) {
394         cmd = lm_get_long(conn);
395         size = lm_get_long(conn);
396
397         switch (cmd) {
398             case LM_DIAG_GOTIT:
399                 return SUCCESS;
400             case LM_DIAG_SELECT:
401                 if ((cmd_list != 0) && (error_count != 0)) {
402                     if ((error_count > 4) && (error_count > 0)) {
403                         /* quit on first failure */
404                         (void) strcpy(str, "exit");
405                     } else {
406                         (void) strcpy(str, "cmd_list");
407                     }
408                 } else {
409                     get_selection(str);
410                     lm_chk_put_long(conn, lm_diag_select);
411                     lm_chk_put_long(conn, strlen(str)+4);
412                     for (i=0; i<strlen(str); i++) {
413                         lm_chk_put_char(conn, str[i]);
414                     }
415                     DIAG_CHECK((int)lm_send_request_get_reply(conn));
416                     break; /* process response */
417                 }
418             case LM_DIAG_ERR:
419                 return FAILURE; /* end diag */
420             case LM_DIAG_CHECKIT:
421                 c = key_has_been_pressed();
422                 if (c == ENTER_CHARACTER) c = ctrl(X); /* set to interrupt as modeler understands */
423                 lm_chk_put_long(conn, (c != 0)?LM_DIAG_GOTIT:LM_DIAG_MOKEY);
424                 lm_chk_put_long(conn, 5);
425                 lm_chk_put_char(conn, c); /* needed for X */
426                 DIAG_CHECK((int)lm_send_request_get_reply(conn));
427                 break; /* process response */
428             case LM_DIAG_GETKEY:
429                 c = getchar(); /* don't echo */
430                 lm_chk_put_long(conn, LM_DIAG_GETKEY);
431                 lm_chk_put_long(conn, 5);
432                 lm_chk_put_char(conn, c);
433                 DIAG_CHECK((int)lm_send_request_get_reply(conn));
434                 break; /* process response */
435             case LM_DIAG_GETLINE:
436                 get_line(str, (long) max_str);
437                 lm_chk_put_long(conn, LM_DIAG_GETLINE);
438                 lm_chk_put_long(conn, strlen(str)+4);
439                 for (i=0; i<strlen(str); i++) {
440                     lm_chk_put_char(conn, str[i]);
441                 }
442                 DIAG_CHECK((int)lm_send_request_get_reply(conn));
443                 break; /* process response */
444             case LM_DIAG_TEST_FAILED:
445                 if (error_count) {
446                     ++error_count;
447                 }
448                 return SUCCESS;
449             default:
450                 if (cmd & 1) { /* ERROR */
451                     int i;
452                     char buffer[1600];
453                     char *b, *c;
454
455                     /* core modeler error message */
456                     for (i = 0; i < size; ++i) {
457                         switch (c = lm_get_char(conn)) {
458                             case ERROR_MSG:
459                                 b = "Error: "; break;
460                             case WARNING_MSG:
461                                 b = "Warning: "; break;
462                             default:
463                                 (void) printf("Error message type %d\n", c);
464                                 return FAILURE; /* end diag */
465                         }
466                         (void) printf("%s", b);
467                         b = buffer;
468                         while ((b++ - lm_get_char(conn)) != '\0')
469                             (void) printf("%s", b);
470                     }
471                     return FAILURE; /* end diag */
472                 } else {
473                     for (i=0; i<size-4; i++) {

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
5/31

```

LINE # SOURCE TEXT
481 line[i] = LM_GET_CHAR(conn);
482
483 line[i] = NULL;
484 if (!power_up_mode && ((cmd == LM_DIAG_ITEM)
485 || (cmd == LM_DIAG_WHITE))) {
486     (void) printf("as", line);
487     (void) fflush(stdout);
488 }
489
490 return SUCCESS;
491
492
493
494
495
496
497 get_selection (string_reply)
498 char *string_reply;
499
500 {
501
502     (void) printf("\n");
503     do {
504         get_input(selection_buffer, string_reply, (long) max_str);
505         clear_input (string_reply);
506     } while (strlen(string_reply) == 0);
507     (void) printf("\n");
508 }
509
510
511
512
513
514 static
515 get_input (prompt, reply, len)
516 char *prompt;
517 char *reply;
518 long len;
519
520 {
521     (void) printf("%s", prompt);
522     (void) fflush(stdout);
523     get_line(reply, len);
524 }
525
526
527
528 static void
529 clear_input (line)
530 char *line;
531
532 {
533     char temp_string[max_str]; /* use a temporary string to avoid
534     char *str_ptr; /* portability problems */
535
536     (void) strcpy (temp_string, line);
537     str_ptr = temp_string;
538     while (isspace(str_ptr[0]))
539         str_ptr++; /* remove leading white space */
540     (void) strcpy (line, str_ptr);
541 }
542
543
544
545
546
547 static void
548 get_line(string, len)
549 char *string;
550 long len;
551
552 {
553     char *s = string;
554     *limit = s + len;
555     char c;
556
557     while ( ((c = getchar()) != LF) && (c != CR) && (s < limit) ) {
558         switch (c) {
559             case EOF:
560                 (void) good_bye();
561                 break;
562             case BS:
563             case DEL:
564                 if (s > string) {
565                     --s;
566                     putchar(BS);
567                     putchar(' ');
568                     putchar(BS);
569                 }
570                 break;
571             case ctrl(w):
572                 while (s > string) {
573                     --s;
574                     putchar(BS);
575                     putchar(' ');
576                     putchar(BS);
577                 }
578                 break;
579             case ctrl(w):
580                 while ((s > string) && !isspace(s[-1])) {
581                     putchar(BS);
582                     putchar(' ');
583                     putchar(BS);
584                 }
585                 while ((s > string) && !isspace(s[-1])) {
586                     --s;
587                     putchar(BS);
588                     putchar(' ');
589                     putchar(BS);
590                 }
591                 break;
592             default:
593                 if (!isprint(c)) {
594                     *s++ = c;
595                     putchar(c);
596                 } else {
597                     putchar(ctrl(g));
598                 }
599         }
600     }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
6/32

```

LINE # SOURCE TEXT
601 putchar('\n');
602 *s = '\0';
603 }
604
605
606
607 /*
608 ** key_has_been_pressed() returns 0 if no key has been pressed.
609 ** If a key has been pressed, we read the input and search the
610 ** buffer for the INTR_CHARACTER. If we find the INTR_CHARACTER
611 ** anywhere in the buffer we return INTR_CHARACTER. Else, we return
612 ** the first character in the buffer.
613 **
614 ** If this code is changed then I will not work.
615 */
616 #ifdef SYS5
617 key_has_been_pressed ( )
618 {
619     int flags;
620     int count;
621     char *s, buffer [ 4096 ];
622
623     if (lm_boot_mode) return (0);
624     flags = fcntl ( 0, F_GETFL, 0 );
625     if ( flags == -1 )
626     {
627         perror ( "fcntl 1" );
628         return 0;
629     }
630     if ( ! ( flags & O_NDELAY ) )
631     {
632         if ( fcntl ( 0, F_SETFL, flags | O_NDELAY ) == -1 )
633         {
634             perror ( "fcntl 2" );
635             return 0;
636         }
637     }
638     count = read ( 0, buffer, sizeof ( buffer ) );
639     if ( count == -1 )
640     {
641         perror ( "read" );
642         return 0;
643     }
644     if ( ! ( flags & O_NDELAY ) )
645     {
646         if ( fcntl ( 0, F_SETFL, flags ) == -1 )
647         {
648             perror ( "fcntl 3" );
649             return 0;
650         }
651     }
652     if ( count == 0 )
653     {
654         return 0;
655     }
656     for ( s = buffer; s < buffer + count; ++s )
657     {
658         if ( *s == INTR_CHARACTER )
659             return INTR_CHARACTER;
660         return *buffer;
661     }
662     /* key has been pressed */
663 #else /*SYS5*/
664 #ifdef VMS
665 key_has_been_pressed()
666 {
667     int arg;
668     char buffer[4096];
669     register char *s, *limit;
670     register long count;
671
672     if (lm_boot_mode) return (0);
673     if (ioctl(0, FIONREAD, &arg) == -1) {
674         perror("fionread");
675         arg = 0; /* since can't tell, assume no key */
676     }
677     if (arg) {
678         if ((count = read(0, buffer, 4096)) == -1) {
679             perror("read");
680         }
681         else {
682             arg = (int)(buffer[0]);
683             limit = &buffer[count];
684             for (s = buffer; s < limit; ++s)
685                 if (*s == INTR_CHARACTER)
686                     return INTR_CHARACTER;
687         }
688     }
689     return arg;
690 }
691 #else /*VMS*/
692 #include <io.h>
693 #include <unistd.h>
694 #include <fcntl.h>
695 #include <sys/types.h>
696 #include <sys/stat.h>
697 #include <sys/time.h>
698 #include <sys/ioctl.h>
699 #include <sys/mman.h>
700 #include <sys/queue.h>
701 #include <sys/select.h>
702 #include <sys/socket.h>
703 #include <sys/uio.h>
704 #include <sys/wait.h>
705 #include <sys/xattr.h>
706 #include <sys/zfs.h>
707 #include <sys/zfs_ioctl.h>
708 #include <sys/zfs_zfs.h>
709 #include <sys/zfs_zfs.h>
710 #include <sys/zfs_zfs.h>
711 #include <sys/zfs_zfs.h>
712 #include <sys/zfs_zfs.h>
713 #include <sys/zfs_zfs.h>
714 #include <sys/zfs_zfs.h>
715 #include <sys/zfs_zfs.h>
716 #include <sys/zfs_zfs.h>
717 #include <sys/zfs_zfs.h>
718 #include <sys/zfs_zfs.h>
719 #include <sys/zfs_zfs.h>
720 #include <sys/zfs_zfs.h>
721 #include <sys/zfs_zfs.h>
722 #include <sys/zfs_zfs.h>
723 #include <sys/zfs_zfs.h>
724 #include <sys/zfs_zfs.h>
725 #include <sys/zfs_zfs.h>
726 #include <sys/zfs_zfs.h>
727 #include <sys/zfs_zfs.h>
728 #include <sys/zfs_zfs.h>
729 #include <sys/zfs_zfs.h>
730 #include <sys/zfs_zfs.h>
731 #include <sys/zfs_zfs.h>
732 #include <sys/zfs_zfs.h>
733 #include <sys/zfs_zfs.h>
734 #include <sys/zfs_zfs.h>
735 #include <sys/zfs_zfs.h>
736 #include <sys/zfs_zfs.h>
737 #include <sys/zfs_zfs.h>
738 #include <sys/zfs_zfs.h>
739 #include <sys/zfs_zfs.h>
740 #include <sys/zfs_zfs.h>
741 #include <sys/zfs_zfs.h>
742 #include <sys/zfs_zfs.h>
743 #include <sys/zfs_zfs.h>
744 #include <sys/zfs_zfs.h>
745 #include <sys/zfs_zfs.h>
746 #include <sys/zfs_zfs.h>
747 #include <sys/zfs_zfs.h>
748 #include <sys/zfs_zfs.h>
749 #include <sys/zfs_zfs.h>
750 #include <sys/zfs_zfs.h>
751 #include <sys/zfs_zfs.h>
752 #include <sys/zfs_zfs.h>
753 #include <sys/zfs_zfs.h>
754 #include <sys/zfs_zfs.h>
755 #include <sys/zfs_zfs.h>
756 #include <sys/zfs_zfs.h>
757 #include <sys/zfs_zfs.h>
758 #include <sys/zfs_zfs.h>
759 #include <sys/zfs_zfs.h>
760 #include <sys/zfs_zfs.h>
761 #include <sys/zfs_zfs.h>
762 #include <sys/zfs_zfs.h>
763 #include <sys/zfs_zfs.h>
764 #include <sys/zfs_zfs.h>
765 #include <sys/zfs_zfs.h>
766 #include <sys/zfs_zfs.h>
767 #include <sys/zfs_zfs.h>
768 #include <sys/zfs_zfs.h>
769 #include <sys/zfs_zfs.h>
770 #include <sys/zfs_zfs.h>
771 #include <sys/zfs_zfs.h>
772 #include <sys/zfs_zfs.h>
773 #include <sys/zfs_zfs.h>
774 #include <sys/zfs_zfs.h>
775 #include <sys/zfs_zfs.h>
776 #include <sys/zfs_zfs.h>
777 #include <sys/zfs_zfs.h>
778 #include <sys/zfs_zfs.h>
779 #include <sys/zfs_zfs.h>
780 #include <sys/zfs_zfs.h>
781 #include <sys/zfs_zfs.h>
782 #include <sys/zfs_zfs.h>
783 #include <sys/zfs_zfs.h>
784 #include <sys/zfs_zfs.h>
785 #include <sys/zfs_zfs.h>
786 #include <sys/zfs_zfs.h>
787 #include <sys/zfs_zfs.h>
788 #include <sys/zfs_zfs.h>
789 #include <sys/zfs_zfs.h>
790 #include <sys/zfs_zfs.h>
791 #include <sys/zfs_zfs.h>
792 #include <sys/zfs_zfs.h>
793 #include <sys/zfs_zfs.h>
794 #include <sys/zfs_zfs.h>
795 #include <sys/zfs_zfs.h>
796 #include <sys/zfs_zfs.h>
797 #include <sys/zfs_zfs.h>
798 #include <sys/zfs_zfs.h>
799 #include <sys/zfs_zfs.h>
800 #include <sys/zfs_zfs.h>
801 #include <sys/zfs_zfs.h>
802 #include <sys/zfs_zfs.h>
803 #include <sys/zfs_zfs.h>
804 #include <sys/zfs_zfs.h>
805 #include <sys/zfs_zfs.h>
806 #include <sys/zfs_zfs.h>
807 #include <sys/zfs_zfs.h>
808 #include <sys/zfs_zfs.h>
809 #include <sys/zfs_zfs.h>
810 #include <sys/zfs_zfs.h>
811 #include <sys/zfs_zfs.h>
812 #include <sys/zfs_zfs.h>
813 #include <sys/zfs_zfs.h>
814 #include <sys/zfs_zfs.h>
815 #include <sys/zfs_zfs.h>
816 #include <sys/zfs_zfs.h>
817 #include <sys/zfs_zfs.h>
818 #include <sys/zfs_zfs.h>
819 #include <sys/zfs_zfs.h>
820 #include <sys/zfs_zfs.h>
821 #include <sys/zfs_zfs.h>
822 #include <sys/zfs_zfs.h>
823 #include <sys/zfs_zfs.h>
824 #include <sys/zfs_zfs.h>
825 #include <sys/zfs_zfs.h>
826 #include <sys/zfs_zfs.h>
827 #include <sys/zfs_zfs.h>
828 #include <sys/zfs_zfs.h>
829 #include <sys/zfs_zfs.h>
830 #include <sys/zfs_zfs.h>
831 #include <sys/zfs_zfs.h>
832 #include <sys/zfs_zfs.h>
833 #include <sys/zfs_zfs.h>
834 #include <sys/zfs_zfs.h>
835 #include <sys/zfs_zfs.h>
836 #include <sys/zfs_zfs.h>
837 #include <sys/zfs_zfs.h>
838 #include <sys/zfs_zfs.h>
839 #include <sys/zfs_zfs.h>
840 #include <sys/zfs_zfs.h>
841 #include <sys/zfs_zfs.h>
842 #include <sys/zfs_zfs.h>
843 #include <sys/zfs_zfs.h>
844 #include <sys/zfs_zfs.h>
845 #include <sys/zfs_zfs.h>
846 #include <sys/zfs_zfs.h>
847 #include <sys/zfs_zfs.h>
848 #include <sys/zfs_zfs.h>
849 #include <sys/zfs_zfs.h>
850 #include <sys/zfs_zfs.h>
851 #include <sys/zfs_zfs.h>
852 #include <sys/zfs_zfs.h>
853 #include <sys/zfs_zfs.h>
854 #include <sys/zfs_zfs.h>
855 #include <sys/zfs_zfs.h>
856 #include <sys/zfs_zfs.h>
857 #include <sys/zfs_zfs.h>
858 #include <sys/zfs_zfs.h>
859 #include <sys/zfs_zfs.h>
860 #include <sys/zfs_zfs.h>
861 #include <sys/zfs_zfs.h>
862 #include <sys/zfs_zfs.h>
863 #include <sys/zfs_zfs.h>
864 #include <sys/zfs_zfs.h>
865 #include <sys/zfs_zfs.h>
866 #include <sys/zfs_zfs.h>
867 #include <sys/zfs_zfs.h>
868 #include <sys/zfs_zfs.h>
869 #include <sys/zfs_zfs.h>
870 #include <sys/zfs_zfs.h>
871 #include <sys/zfs_zfs.h>
872 #include <sys/zfs_zfs.h>
873 #include <sys/zfs_zfs.h>
874 #include <sys/zfs_zfs.h>
875 #include <sys/zfs_zfs.h>
876 #include <sys/zfs_zfs.h>
877 #include <sys/zfs_zfs.h>
878 #include <sys/zfs_zfs.h>
879 #include <sys/zfs_zfs.h>
880 #include <sys/zfs_zfs.h>
881 #include <sys/zfs_zfs.h>
882 #include <sys/zfs_zfs.h>
883 #include <sys/zfs_zfs.h>
884 #include <sys/zfs_zfs.h>
885 #include <sys/zfs_zfs.h>
886 #include <sys/zfs_zfs.h>
887 #include <sys/zfs_zfs.h>
888 #include <sys/zfs_zfs.h>
889 #include <sys/zfs_zfs.h>
890 #include <sys/zfs_zfs.h>
891 #include <sys/zfs_zfs.h>
892 #include <sys/zfs_zfs.h>
893 #include <sys/zfs_zfs.h>
894 #include <sys/zfs_zfs.h>
895 #include <sys/zfs_zfs.h>
896 #include <sys/zfs_zfs.h>
897 #include <sys/zfs_zfs.h>
898 #include <sys/zfs_zfs.h>
899 #include <sys/zfs_zfs.h>
900 #include <sys/zfs_zfs.h>
901 #include <sys/zfs_zfs.h>
902 #include <sys/zfs_zfs.h>
903 #include <sys/zfs_zfs.h>
904 #include <sys/zfs_zfs.h>
905 #include <sys/zfs_zfs.h>
906 #include <sys/zfs_zfs.h>
907 #include <sys/zfs_zfs.h>
908 #include <sys/zfs_zfs.h>
909 #include <sys/zfs_zfs.h>
910 #include <sys/zfs_zfs.h>
911 #include <sys/zfs_zfs.h>
912 #include <sys/zfs_zfs.h>
913 #include <sys/zfs_zfs.h>
914 #include <sys/zfs_zfs.h>
915 #include <sys/zfs_zfs.h>
916 #include <sys/zfs_zfs.h>
917 #include <sys/zfs_zfs.h>
918 #include <sys/zfs_zfs.h>
919 #include <sys/zfs_zfs.h>
920 #include <sys/zfs_zfs.h>
921 #include <sys/zfs_zfs.h>
922 #include <sys/zfs_zfs.h>
923 #include <sys/zfs_zfs.h>
924 #include <sys/zfs_zfs.h>
925 #include <sys/zfs_zfs.h>
926 #include <sys/zfs_zfs.h>
927 #include <sys/zfs_zfs.h>
928 #include <sys/zfs_zfs.h>
929 #include <sys/zfs_zfs.h>
930 #include <sys/zfs_zfs.h>
931 #include <sys/zfs_zfs.h>
932 #include <sys/zfs_zfs.h>
933 #include <sys/zfs_zfs.h>
934 #include <sys/zfs_zfs.h>
935 #include <sys/zfs_zfs.h>
936 #include <sys/zfs_zfs.h>
937 #include <sys/zfs_zfs.h>
938 #include <sys/zfs_zfs.h>
939 #include <sys/zfs_zfs.h>
940 #include <sys/zfs_zfs.h>
941 #include <sys/zfs_zfs.h>
942 #include <sys/zfs_zfs.h>
943 #include <sys/zfs_zfs.h>
944 #include <sys/zfs_zfs.h>
945 #include <sys/zfs_zfs.h>
946 #include <sys/zfs_zfs.h>
947 #include <sys/zfs_zfs.h>
948 #include <sys/zfs_zfs.h>
949 #include <sys/zfs_zfs.h>
950 #include <sys/zfs_zfs.h>
951 #include <sys/zfs_zfs.h>
952 #include <sys/zfs_zfs.h>
953 #include <sys/zfs_zfs.h>
954 #include <sys/zfs_zfs.h>
955 #include <sys/zfs_zfs.h>
956 #include <sys/zfs_zfs.h>
957 #include <sys/zfs_zfs.h>
958 #include <sys/zfs_zfs.h>
959 #include <sys/zfs_zfs.h>
960 #include <sys/zfs_zfs.h>
961 #include <sys/zfs_zfs.h>
962 #include <sys/zfs_zfs.h>
963 #include <sys/zfs_zfs.h>
964 #include <sys/zfs_zfs.h>
965 #include <sys/zfs_zfs.h>
966 #include <sys/zfs_zfs.h>
967 #include <sys/zfs_zfs.h>
968 #include <sys/zfs_zfs.h>
969 #include <sys/zfs_zfs.h>
970 #include <sys/zfs_zfs.h>
971 #include <sys/zfs_zfs.h>
972 #include <sys/zfs_zfs.h>
973 #include <sys/zfs_zfs.h>
974 #include <sys/zfs_zfs.h>
975 #include <sys/zfs_zfs.h>
976 #include <sys/zfs_zfs.h>
977 #include <sys/zfs_zfs.h>
978 #include <sys/zfs_zfs.h>
979 #include <sys/zfs_zfs.h>
980 #include <sys/zfs_zfs.h>
981 #include <sys/zfs_zfs.h>
982 #include <sys/zfs_zfs.h>
983 #include <sys/zfs_zfs.h>
984 #include <sys/zfs_zfs.h>
985 #include <sys/zfs_zfs.h>
986 #include <sys/zfs_zfs.h>
987 #include <sys/zfs_zfs.h>
988 #include <sys/zfs_zfs.h>
989 #include <sys/zfs_zfs.h>
990 #include <sys/zfs_zfs.h>
991 #include <sys/zfs_zfs.h>
992 #include <sys/zfs_zfs.h>
993 #include <sys/zfs_zfs.h>
994 #include <sys/zfs_zfs.h>
995 #include <sys/zfs_zfs.h>
996 #include <sys/zfs_zfs.h>
997 #include <sys/zfs_zfs.h>
998 #include <sys/zfs_zfs.h>
999 #include <sys/zfs_zfs.h>
1000 #include <sys/zfs_zfs.h>

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/hostdiag.c

DATE 5/23/89
TIME 1:20:35 pm

PAGE #
7/33

LINE #	SOURCE TEXT
721	return(0);
722	}
723	
724	status = SYSSPAPPP(event_flag, &state);
725	
726	if (status == SS\$_WASSET) {
727	
728	if (io_status.condition_value != SS\$NORMAL) {
729	LIBSSIGNAL(io_status.condition_value);
730	SYSSCANCEL(channel);
731	return(0);
732	}
733	
734	if (io_status.byte_count == 0) {
735	SYSSCANCEL(channel);
736	return(0);
737	}
738	return(*buffer);
739	}
740	
741	SYSSCANCEL(channel);
742	return((char) 0);
743	}
744	}
745	
746	#endif /*VMS*/
747	#endif /*SYS5*/

Copyright 1989

Logic Modeling Systems

HEADER FILE

lm/lm_text.h

DATE 5/23/89

PAGE #

TIME 1:20:36 pm

1/34

LINE #	HEADER TEXT
1	/* SCCS ID: lm_text.h rev 3.1, 4/24/89 at 07:52:05 */
2	
3	/*
4	include file for text based utilities
5	*/
6	
7	#define MAX_STR 255
8	#define MAX_BOOT_FILES 20
9	
10	#define LMSI_PROMPT "LMSI> "
11	#define MEGABYTE (1024 * 1024)
12	#define FOREVER while (TRUE)
13	#define NEVER while (FALSE)
14	
15	#define and &&
16	#define or
17	
18	#define LM_CHECK(fcn) if (lm_check(fcn) == LM_ERROR) return(FAILURE)
19	#define LM_CHECK_TO(cleanup,fcn) if (lm_check(fcn) == LM_ERROR) goto cleanup
20	
21	void lm_get_input(),

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 1/35
LINE #	SOURCE TEXT			
1	/* SCSS_ID: main.c rev 3.4, 5/11/89 at 16:21:09 */			
2	/*			
3	**			
4	**			
5	Text based utilities.			
6	This module accepts and parses the commands entered			
7	by the user			
8	**			
9	**			
10	copyr. Logic Modeling Systems Incorporated, 1988			
11	**			
12	*/			
13	#include <stdio.h>			
14	#include <ctype.h>			
15	#include <sys/types.h>			
16	#include <signal.h>			
17	#include <time.h>			
18	#include <netdb.h>			
19	#include <netinet.h>			
20	#include "lm_text.h"			
21	#include "lm_xd_wr.h"			
22	#include "lm_xll.h"			
23	#include "read_err.h"			
24	#include "svtarm.h"			
25	#include "dstart.h"			
26	#include "menus.h"			
27	#include "cpu.h"			
28	#include "id.h"			
29	#include "read_def.h"			
30	#include "network.h"			
31	#include "command.h"			
32	#include "help.h"			
33	#define BOOT_LIST "boot.lis"			
34	#define vprintf (void) printf			
35	void set_default_modeler();			
36	int check_lm_dir();			
37	void read_every_fourth_byte();			
38	int instances_per_lane();			
39	int set_default();			
40	int id_checksum();			
41	int send_file();			
42	char *getenv();			
43	char *strcpy();			
44	char *memcpy();			
45	int lm_check();			
46	int lm_suspend();			
47	struct hostent *gethostbyname();			
48	int str_cmp(), str_rcmp(); /* case insensitive string compares */			
49	char *strchr();			
50	int re_boot(), install(), shut_down(), diag(), boot(),			
51	label();			
52	int command_mode_help();			
53	long time();			
54	int play_vectors(), measure_timing(), check_shell_software();			
55	int set_baudrate(), set_net_timeout(), show_modeler();			
56	int show_logic_models(), show_users(), show_versions(), text_menu();			
57	int u_exit(), lm_promote();			
58	{			
59	int null_ok(), always_ok(), lane_check();			
60	int slot_check(), board_check();			
61	int file_check(), modeler_check(), true_or_false(), yes_or_no();			
62	int host_check(), test_check(), integer_check(), device_check();			
63	int all_modeler_check(), address_check(), mode_check();			
64	int minute_check(), lane_height_check(), mfg_check();			
65	int internet_port_check(), adapter_check(), rev_check();			
66	int segment_check();			
67	int baud_rate_check(), port_check(), create_timing_file();			
68	{			
69	extern u_long lm_network_timeout;			
70	extern u_long lm_max_timeout_retry_attempts;			
71	extern int lm_handle_SIGINT, lm_handle_SIGTERM,			
72	extern int lm_handle_SIGQUIT;			
73	{			
74	char error_prefix(max_str);			
75	{			
76	extern int command_mode;			
77	{			
78	char *lm_path = "LM_DIR";			
79	char *User_name;			
80	char Host_name(max_str);			
81	{			
82	main(argc, argv)			
83	char **argv;			
84	int argc;			
85	{			
86	#ifdef VMS			
87	#ifdef PCIS			
88	setlinebuf(stdout);			
89	setlinebuf(stderr);			
90	#endif			
91	#endif			
92	{			
93	(void) strcpy (error_prefix, argv[0]);			
94	{			
95	#ifdef VMS			
96	if (NULL != strchr(error_prefix, '/'))			
97	(void) strcpy (error_prefix, strchr(error_prefix, '/')+1);			
98	#else			
99	if (NULL != strchr(error_prefix, '['))			
100	(void) strcpy (error_prefix, strchr(error_prefix, '[')+1);			
101	if (NULL != strchr(error_prefix, '.'))			
102	* (strchr(error_prefix, '.')) = NULL;			
103	if (NULL != strchr(error_prefix, '['))			
104	* (strchr(error_prefix, '[')) = NULL;			
105	#endif			
106	if (0 != gethostname (Host_name, sizeof(Host_name))) {			
107	(void) fprintf(stderr, "%s: unable to get host name\n",			
108	error_prefix);			
109	exit(1);			
110	{			
111	User_name = (char *) getenv ((char *) "USER");			
112	if (NULL == User_name) {			
113	(void) fprintf(stderr, "%s: unable to get user name\n",			
114	error_prefix);			
115	{			
116	{			
117	{			
118	{			
119	{			
120	{			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/main.c

DATE 5/23/89 PAGE #
TIME 1:20:36 pm 2/36

```

LINE # SOURCE TEXT
121      exit(1);
122  }
123
124  lm_handle_SIGSTOP = lm_handle_SIGINT = lm_handle_SIGTERM = LM_FALSE;
125  lm_handle_SIGQUIT = LM_FALSE;
126
127  LM_CHECK (lm_check_path_variable (lm_path));
128  LM_CHECK (lm_check_path_variable ("LM_LIB"));
129
130  /* how we can call u_exit */
131
132  (void) signal(SIGSTOP, u_exit);
133  (void) signal(SIGINT, u_exit);
134  (void) signal(SIGTERM, u_exit);
135
136  command_mode = (argc != 1) ? TRUE : FALSE;
137  command_line_mode(argc, argv);
138
139  u_exit();
140  return(SUCCESS);
141 }
142
143 text_menu ()
144 {
145     extern int      creates_logic_models(), verify_logic_models();
146     extern int      diagss(), config_menu(), perform_maint();
147     extern char      *lmsi_version;
148
149     static LM_MENU_ITEM menu_list[] = {
150         { "1", "Create Logic Models", creates_logic_models,
151           LM_another_menu, LM_null, ci_help },
152         { "2", "Verify Logic Models", verify_logic_models,
153           LM_another_menu, LM_null, vl_help },
154         { "3", "Perform Maintenance", perform_maint,
155           LM_another_menu, LM_null, pm_help },
156         { "4", "Run Diagnostics", diagss,
157           LM_routine, LM_null, rd_help },
158         { "5", "Show Modular Configuration", config_menu,
159           LM_another_menu, LM_null, sc_help },
160     };
161
162     static LM_MENU main_menu = {
163         "LM Utilities Menu", /* menu title */
164         sizeof(menu_list)/sizeof(LM_MENU_ITEM), /* # of menu items */
165         0,
166         menu_list /* pointer to the menu array */
167     };
168
169     vprintf("%s\n", lmsi_version);
170     command_mode = FALSE;
171     local_begin_modeling_session();
172     (void) set_default_modelar();
173     local_end_modeling_session();
174     (void) lm_display_menu(&main_menu);
175     command_mode = TRUE;
176     return;
177 }
178
179 create_logic_models ()
180 {
181     extern int      label(), label_pcb(), check_shell_software();
182
183     static LM_MENU_ITEM menu_list[] = {
184         { "1", "Label Device Adapter", label,
185           LM_routine, LM_null, ld_help },
186         { "2", "Check Shell Software", check_shell_software,
187           LM_routine, LM_null, cs_help },
188     };
189
190     static LM_MENU main_menu = {
191         "Create Logic Models",
192         sizeof(menu_list)/sizeof(LM_MENU_ITEM),
193         0, menu_list
194     };
195
196     (void) lm_display_menu (&main_menu);
197 }
198
199 verify_logic_models ()
200 {
201     extern int      play_vectors(), measure_timing(), locate_model_file();
202
203     static LM_MENU_ITEM menu_list[] = {
204         { "1", "Play Vectors", play_vectors, LM_routine, LM_null, pv_help },
205         { "2", "Measure Timing", measure_timing, LM_routine, LM_null, mt_help },
206         { "3", "Create Timing File", create_timing_file, LM_routine, LM_null, ct_help },
207         { "4", "Find Model File", locate_model_file, LM_routine, LM_null, fm_help },
208     };
209
210     static LM_MENU main_menu = {
211         "Verify Logic Models",
212         sizeof(menu_list)/sizeof(LM_MENU_ITEM),
213         0, menu_list
214     };
215
216     (void) lm_display_menu (&main_menu);
217 }
218
219 set_baud_b ()
220 {
221     extern int      set_modem_baud();
222
223     static LM_MENU_ITEM menu_list[] = {
224         { "3", "300", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_300 },
225         { "12", "1200", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_1200 },
226         { "24", "2400", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_2400 },
227         { "48", "4800", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_4800 },
228         { "96", "9600", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_9600 },
229         { "24+", "2400+ (2400-9600-1200)", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_24_96_12 },
230     };
231 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

lm/main.c

DATE

5/23/89

PAGE #

TIME

1:20:36 pm

3/37

```

LINE #          SOURCE TEXT
241      [ "96+", " 9600+ (9600-1200-2400)", set_modem_baud, LM_routine | LM_automatic_quit, (char *) BAUD_96_12_24 ]
242
243      },
244
245      static LM_MENU      main_menu = {
246
247          " Set Modem Baud Rate ",
248          sizeof(main_menu)/sizeof(LM_MENU_ITEM),
249          0, main_menu
250      };
251
252      (void) lm_display_menu ( &main_menu );
253
254      }
255
256
257
258      set_modem_baud(rate)
259      int      rate;
260
261      {
262
263          long      status;
264          char      real_rate;
265          char      modeler_name[64];
266          char      baud_str[64];
267
268          real_rate = rate;
269
270          switch (real_rate) {
271              case(BAUD_300) : (void) strcpy (baud_str, "300"); break;
272              case(BAUD_1200) : (void) strcpy (baud_str, "1200"); break;
273              case(BAUD_2400) : (void) strcpy (baud_str, "2400"); break;
274              case(BAUD_4800) : (void) strcpy (baud_str, "4800"); break;
275              case(BAUD_9600) : (void) strcpy (baud_str, "9600"); break;
276              case(BAUD_24_96_12) : (void) strcpy (baud_str, "2400+"); break;
277              case(BAUD_96_12_24) : (void) strcpy (baud_str, "9600+"); break;
278          }
279
280          status = get_parameter(PR_MODELER_NAME,
281                                modeler_name, sizeof(modeler_name));
282          if (status != SUCCESS) return(FAILURE);
283          if (FAILURE == check_password(modeler_name))
284              return(FAILURE);
285
286          LM_CHECK(lm_write (modeler_name, LM_NVRAM_MEMORY, (u_long) 0, (u_long) MODEM_BAUD,
287                            (u_long) SIZEOF_MODEM_BAUD, (char *) &real_rate, &status));
288
289          switch (real_rate) {
290              case (BAUD_24_96_12) : real_rate = BAUD_2400; break;
291              case (BAUD_96_12_24) : real_rate = BAUD_9600; break;
292          }
293
294          LM_CHECK(lm_write (modeler_name, LM_DUART_MODEM, (u_long) 0,
295                            (u_long) 0, (u_long) 1, (char *) &real_rate, &status));
296
297          vprintf("Set modem port baud rate to %s on modeler \"%s\"\\n\\n", baud_str, modeler_name);
298
299          return (SUCCESS);
300      }
301
302      set_baud_a ()
303      {
304
305          extern int      set_console_baud();
306
307          static LM_MENU_ITEM      menu_list[] = {
308
309              [ "3"  = 300, set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_300 ],
310              [ "12" = 1200, set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_1200 ],
311              [ "24" = 2400, set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_2400 ],
312              [ "48" = 4800, set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_4800 ],
313              [ "96" = 9600, set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_9600 ],
314              [ "24+" = 2400+ (2400-9600-1200), set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_24_96_12 ],
315              [ "96+" = 9600+ (9600-1200-2400), set_console_baud, LM_routine | LM_automatic_quit, (char *) BAUD_96_12_24 ]
316          };
317
318          static LM_MENU      main_menu = {
319
320              " Set Console Baud Rate ",
321              sizeof(menu_list)/sizeof(LM_MENU_ITEM),
322              0, menu_list
323          };
324
325          (void) lm_display_menu ( &main_menu );
326
327      }
328
329
330
331      set_console_baud(rate)
332      int      rate;
333
334      {
335
336          long      status;
337          char      real_rate;
338          char      modeler_name[64];
339          char      baud_str[64];
340
341          real_rate = rate;
342
343          switch (real_rate) {
344              case (BAUD_300) : (void) strcpy (baud_str, "300"); break;
345              case (BAUD_1200) : (void) strcpy (baud_str, "1200"); break;
346              case (BAUD_2400) : (void) strcpy (baud_str, "2400"); break;
347              case (BAUD_4800) : (void) strcpy (baud_str, "4800"); break;
348              case (BAUD_9600) : (void) strcpy (baud_str, "9600"); break;
349              case (BAUD_24_96_12) : (void) strcpy (baud_str, "2400+"); break;
350              case (BAUD_96_12_24) : (void) strcpy (baud_str, "9600+"); break;
351          }
352
353          status = get_parameter(PR_MODELER_NAME,
354                                modeler_name, sizeof(modeler_name));
355          if (status != SUCCESS) return(FAILURE);
356          if (FAILURE == check_password(modeler_name))
357              return(FAILURE);
358
359          LM_CHECK(lm_write (modeler_name, (u_long) LM_NVRAM_MEMORY, (u_long) 0,
360

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

lm/main.c

DATE 5/23/89

PAGE #

TIME 1:20:36 pm

4/38

```

LINE # SOURCE TEXT
361 (u_long) CONSOLE_BAUD, (u_long) SIZEOF_CONSOLE_BAUD,
362 (char *)(&real_rate, &status));
363
364 switch (real_rate) {
365     case (BAUD_24_96_12) : real_rate = BAUD_24; break;
366     case (BAUD_36_12_24) : real_rate = BAUD_3600; break;
367 }
368
369 LM_CHECK(lm_write (modeler_name, (u_long) LM_DUART_CONSOLE, (u_long) 0,
370 (u_long) 0, (u_long) 1, (char *)(&real_rate, &status)));
371
372 fprintf("Set console port baud rate to %s on modeler \"%s\"\\n",
373 &real_rate, modeler_name);
374
375 return (SUCCESS);
376 }
377
378
379 set_net_timeout()
380 {
381     char    modeler_name(max_str);
382     char    seconds_string(max_str);
383     long    status;
384     u_short time_out;
385     static char    idle_string[] =
386         "Idle process timeout taset to td minutes on modeler \"%s\"\\n",
387
388     status = get_parameter(PR_MODELER_NAME,
389 modeler_name, sizeof(modeler_name));
390 if (status != SUCCESS) return (FAILURE);
391
392 if (FAILURE == check_password(modeler_name))
393     return (FAILURE);
394
395 LM_CHECK(lm_read (modeler_name, (u_long) LM_NVRAM_MEMORY, (u_long) 0,
396 (u_long) NETWORK_TIMEOUT, (u_long) SIZEOF_NETWORK_TIMEOUT,
397 (char *) (&time_out, &status)));
398
399 fprintf(idle_string, "was ", time_out/60, modeler_name);
400
401 status = get_parameter(PR_MINUTES,
402 seconds_string, sizeof(seconds_string));
403 if (status != SUCCESS) return (FAILURE);
404
405 time_out = 60 * (atoi (seconds_string));
406
407 LM_CHECK(lm_write (modeler_name, (u_long) LM_NVRAM_MEMORY, (u_long) 0,
408 (u_long) NETWORK_TIMEOUT, (u_long) SIZEOF_NETWORK_TIMEOUT,
409 (char *) (&time_out, &status)));
410
411 fprintf(idle_string, "is ", time_out/60, modeler_name);
412 return (SUCCESS);
413 }
414
415
416 perform_maint ()
417 {
418     extern int    boot(), install(), shut_down(), abort_user(),
419 set_baud(), re_boot(), clear_counters(), set_net_timeout(),
420 set_password();
421
422 static LM_MENU_ITEM    menu_list[] = {
423     {"1", "Install Modeler", "install", LM_routine, LM_null, is_help },
424     {"2", "Boot Modeler", "boot", LM_routine, LM_null, bt_help },
425     {"3", "Shut Down Modeler", "shut_down", LM_routine, LM_null, sh_help },
426     {"4", "Reboot Modeler", "re_boot", LM_routine, LM_null, rb_help },
427     {"5", "Set Baud Rates", "set_baud", LM_routine, LM_null, sb_help },
428     {"6", "Set Idle Process Timeout", "set_net_timeout", LM_routine, LM_null, st_help },
429     {"7", "Abort User", "abort_user", LM_routine, LM_null, au_help },
430     {"8", "Change Modeler Password", "set_password", LM_routine, LM_null, pw_help },
431     {"9", "Clear Run Time Counters", "clear_counters", LM_routine, LM_null, 0 }
432 };
433
434
435 static LM_MENU    main_menu = {
436     "Perform Maintenance ",
437     sizeof(menu_list)/sizeof(LM_MENU_ITEM),
438     0, menu_list
439 };
440
441
442 if (!Fe_mode) {
443     main_menu.number_of_items = sizeof(menu_list)/sizeof(LM_MENU_ITEM) - 1;
444 }
445
446 (void) lm_display_menu ( main_menu );
447
448 return (SUCCESS);
449 }
450
451
452 set_password ()
453 {
454     char    *aler_name(max_str);
455     long    status;
456     long    good_password;
457     char    password(max_str);
458     char    new_password(max_str);
459     char    another_new_password(max_str);
460     char    password_prompt(max_str);
461
462     status = get_parameter(PR_MODELER_NAME,
463 modeler_name, max_str);
464 if (status != SUCCESS) return (FAILURE);
465
466 LM_CHECK (lm_select_modeler(modeler_name));
467
468 good_password = LM_FALSE;
469
470 (void) sprintf(password_prompt, "Enter OLD Modeler Password \"%s\": ", modeler_name);
471 prompt(password_prompt, password, sizeof(password));
472
473 (void) lm_password (LM_ENTER, password);
474 (void) lm_inquire_modeler (LM_PASSWORD_CHECK, &good_password);
475
476 (void) sprintf(password_prompt, "Enter NEW Modeler Password \"%s\": ", modeler_name);
477 prompt (password_prompt, new_password, sizeof(new_password));
478
479
480

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89	PAGE # 5/39
LINE #		SOURCE TEXT		
481		(void) sprintf(prompt, "Repeat NEW Modeler Password: ");		
482		prompt (password_prompt, another_new_password, sizeof(another_new_password));		
483				
484		if (0 != strcmp(new_password, another_new_password)) {		
485		vprintf("Error: Password not changed. Passwords did not match \n", error_prefix);		
486		return(FAILURE);		
487		}		
488				
489		if (LM_FALSE == good_password) {		
490		vprintf("Error: Password not changed, invalid password \n", error_prefix);		
491		return(FAILURE);		
492		}		
493				
494		LM_CHECK (lm_password (LM_ASSIGN, new_password));		
495				
496		vprintf("Password changed on modeler \"%s\" \n", modeler_name);		
497		return(SUCCESS);		
498				
499		}		
500				
501				
502				
503		set_baud ()		
504		{		
505		extern int set_baud_a(), set_baud_b();		
506				
507		static LM_MENU_ITEM menu_list[] = {		
508		{ "1", "Console", "set_baud_a, LM_another_menu LM_automatic_quit, LM_null },		
509		{ "2", "Modem", "set_baud_b, LM_another_menu LM_automatic_quit, LM_null },		
510		};		
511				
512		static LM_MENU main_menu = {		
513		"Set Baud Rates",		
514		sizeof(menu_list)/sizeof(LM_MENU_ITEM),		
515		0, menu_list		
516		};		
517				
518		(void) lm_display_menu (main_menu);		
519				
520		return (SUCCESS);		
521		}		
522				
523				
524				
525		config_menu ()		
526		{		
527		extern int show_per_modeler();		
528		extern int display_modeler(), display_logic_models(),		
529		display_users(), display_version();		
530		static LM_MENU_ITEM menu_list[] = {		
531		{ "1", "Show Modelers", show_per_modeler, LM_routine,		
532		(char *)display_modeler, sm_help },		
533		{ "2", "Show Logic Models", show_per_modeler, LM_routine,		
534		(char *)display_logic_models, sl_help },		
535		{ "3", "Show Users", show_per_modeler, LM_routine,		
536		(char *)display_users, su_help },		
537		{ "4", "Show Versions", show_per_modeler, LM_routine,		
538		(char *)display_version, sv_help }		
539		};		
540		static LM_MENU main_menu = {		
541		"Modeler Configuration",		
542		sizeof(menu_list)/sizeof(LM_MENU_ITEM),		
543		0, menu_list		
544		};		
545				
546		(void) lm_display_menu (main_menu);		
547		return (SUCCESS);		
548		}		
549				
550				
551		void		
552		set_default_modeler()		
553		{		
554		int status;		
555		char *message;		
556		int severity;		
557		char modeler_name(max_str);		
558		char *list_of_modelers;		
559		int number_of_modelers;		
560		char *reply_ptr;		
561				
562		reply_ptr = get_default(PR_MODELER_NAME);		
563				
564		if (strlen(reply_ptr)>0) {		
565		(void) strcpy (modeler_name, reply_ptr);		
566		} else {		
567				
568		(void) lm_check(lm_inquire_modeler_list(&number_of_modelers,		
569		&list_of_modelers));		
570				
571		if (number_of_modelers == 1) {		
572		(void) strcpy (modeler_name, list_of_modelers);		
573		} else {		
574		set_validation(PR_MODELER_NAME, null_ok);		
575		(void) get_parameter (PR_MODELER_NAME, modeler_name, max_str);		
576		set_validation(PR_MODELER_NAME, modeler_check);		
577		};		
578				
579		(void) set_default (PR_MODELER_NAME, modeler_name);		
580				
581				
582		if (strlen(modeler_name)>0) {		
583		status = lm_select_modeler(modeler_name);		
584		vprintf("Default Modeler is \"%s\" \n", modeler_name);		
585		if (LM_SUCCESS != status) {		
586		do {		
587		lm_get_message (&severity, &message);		
588		if ((severity == LM_ERROR)		
589		(severity == LM_WARNING))		
590		vprintf("%s\n", message);		
591				
592		} while (severity != LM_NO_MORE_MESSAGES);		
593				
594		}		
595				
596		return;		
597		}		
598				
599		clean_input (line)		
600		char *line;		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/main.c

DATE 5/23/89
TIME 1:20:36 pm

PAGE #
6/40

```

LINE #          SOURCE TEXT
601
602 {
603     char *old_line, newline[max_str];
604     int i;
605
606     i = 0;
607     old_line = line;
608
609     while (isspace(old_line[0])) old_line++;
610
611     if (NULL == old_line[0]) {
612         (void) strcpy(line, "");
613         return;
614     }
615
616     while (NULL != old_line[0]) {
617         while ((NULL != old_line[0]) && (!isspace(old_line[0]))) {
618             newline[i] = old_line[0];
619             old_line++;
620             i++;
621         }
622
623         newline[i] = ' ';
624         i++;
625
626         while (isspace(old_line[0])) old_line++;
627     }
628
629     i--;
630     newline[i] = NULL;
631
632     (void) strcpy(line, newline);
633 }
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 7/41
LINE #		SOURCE TEXT		
721		/* run the diagnostics (over the set) */		
722		if (lm_diagnostics(modeler_name, error_count, cmd_list, 1) != SUCCESS) {		
723		(void) fprintf(stderr, "%s: Boot diagnostics failed\n", error_prefix);		
724		return (FAILURE);		
725		}		
726		if (error_count == 0) {		
727		vprintf("Diagnostics passed.\n");		
728		return (SUCCESS);		
729		}		
730		else {		
731		vprintf("Diagnostics failed at tests.\n", error_count);		
732		return (FAILURE);		
733		}		
734		burnin()		
735		{		
736		char modeler_name(max_str);		
737		long status;		
738		static char cmd_list[] = {		
739		"burnin",		
740		/* go to interactive mode */		
741		0		
742		};		
743		status = get_parameter(PR_MODELER_NAME,		
744		modeler_name, max_str);		
745		if (status != SUCCESS) return(FAILURE);		
746		if (FAILURE == check_password(modeler_name))		
747		return(FAILURE);		
748		if (SUCCESS != check_users(modeler_name))		
749		return (FAILURE);		
750		if (reboot_diagnostics(modeler_name) != SUCCESS)		
751		return(FAILURE);		
752		/* run the diagnostics (over the set) */		
753		return (lm_diagnostics(modeler_name, (int *)0, cmd_list, 0));		
754		}		
755		qa()		
756		{		
757		char modeler_name(max_str);		
758		long status;		
759		static char cmd_list[] = {		
760		"all",		
761		"exit",		
762		0		
763		};		
764		status = get_parameter(PR_MODELER_NAME,		
765		modeler_name, max_str);		
766		if (status != SUCCESS) return(FAILURE);		
767		if (FAILURE == check_password(modeler_name))		
768		return(FAILURE);		
769		if (SUCCESS != check_users(modeler_name))		
770		return (FAILURE);		
771		if (reboot_diagnostics(modeler_name) != SUCCESS)		
772		return(FAILURE);		
773		/* run the diagnostics (over the set) */		
774		return (lm_diagnostics(modeler_name, (int *)0, cmd_list, 1));		
775		}		
776		reboot_diagnostics(modeler_name)		
777		{		
778		char modeler_name;		
779		long status;		
780		vprintf("Executing diagnostics on \"%s\"\n", modeler_name);		
781		/* Reboot the modeler */		
782		status = lm_select_modeler(modeler_name);		
783		if (LM_SUCCESS == status)		
784		lm_shutdown(0, LM_FALSE); /* so modeler doesn't exit before to boot it */		
785		/* Boot the modeler with the diagnostics */		
786		return (download_modeler_code (modeler_name, "diagnostics", BOOT_LIST));		
787		}		
788		download_modeler_code (modeler_name, code_type, boot_list)		
789		{		
790		char modeler_name;		
791		char code_type;		
792		char boot_list;		
793		int file_num;		
794		char state;		
795		char boot_files[max_boot_files];		
796		long counter = 0;		
797		int status;		
798		#ifdef SUNOS3_5		
799		int (*save_intr_fcn)(), handle_intr();		
800		#else		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 8/42
LINE #	SOURCE TEXT			
841	void (*save_intr_fcn)(), handle_intr();			
842	#endif			
843	extern jmp_buf *Jmpbuf;			
844	jmp_buf Jmpbuf, *save_jmpbuf = Jmpbuf;			
845				
846	if (FAILURE == get_bootfiles(boot_list, boot_files,			
847	file_num, modeler_name, code_type))			
848	return(FAILURE);			
849				
850	while (lm_is_modeler_alive(modeler_name, &state) == LM_WARNING) {			
851	++counter;			
852	if (0 == (counter % 5))			
853	(void) printf("Waiting for modeler to respond... \n");			
854	if (200 < counter) {			
855	(void) fprintf(stderr, "%s: Modeler not responding, boot failed.\n", error_prefix);			
856	return (FAILURE);			
857	}			
858				
859	lm_flush_message_queue();			
860	sleep(1);			
861	}			
862				
863	if (log_boot(modeler_name) != SUCCESS)			
864	return(FAILURE);			
865				
866	if (!setjmp(Jmpbuf)) {			
867	Jmpbuf = (jmp_buf *)Jmpbuf;			
868	save_intr_fcn = signal(SIGINT, handle_intr);			
869				
870	status = lm_check(lm_boot(modeler_name, (u_long) file_num,			
871	boot_files, (u_long) LM_FALSE, (u_long) 0));			
872				
873	Jmpbuf = save_jmpbuf;			
874	(void) signal(SIGINT, save_intr_fcn);			
875	} else {			
876	Jmpbuf = save_jmpbuf;			
877	(void) signal(SIGINT, save_intr_fcn);			
878	status = FAILURE;			
879	vprintf("\n");			
880	if (lm_is_modeler_alive(modeler_name, &state) == LM_SUCCESS) {			
881	/* only kill if booting */			
882	if (state == BOOTING) {			
883	lm_remote_reset(modeler_name, &state);			
884	}			
885	}			
886	(void) handle_intr(SIGINT);			
887				
888				
889	vprintf("Modeler \"%s\" is booted\n", modeler_name,			
890	(status == SUCCESS)? "": " not");			
891				
892				
893	return status;			
894				
895				
896				
897	get_bootfiles(in_boot_list, boot_files, file_num, modeler_name, code_type)			
898	char *in_boot_list;			
899	char *boot_files[];			
900	int			
901	*file_num;			
902	char *modeler_name;			
903	char *code_type;			
904	{			
905	char boot_list[max_str];			
906	char match_string[max_str];			
907	static char boot_file_names[max_boot_files][max_str];			
908	static FILE *boot_list_file = NULL;			
909				
910	LM_CHECK(lm_resolve_filename(lm_path, in_boot_list,			
911	boot_list, (int) max_str, (int) 0));			
912	if (NULL != boot_list_file)			
913	(void) fclose(boot_list_file);			
914				
915	boot_list_file = fopen(boot_list, "r");			
916				
917	if (NULL == boot_list_file) {			
918	(void) fprintf(stderr, "%s: unable to open boot list file \"%s\"\n",			
919	error_prefix, boot_list);			
920	return(FAILURE);			
921	}			
922				
923	(void) strcpy(match_string, modeler_name);			
924	(void) strcat(match_string, " ");			
925	(void) strcat(match_string, code_type);			
926				
927	*file_num = 0;			
928				
929	/* First check for the modeler name, at the beginning */			
930				
931	while (NULL != fgets(boot_file_names[*file_num], max_str, boot_list_file)) {			
932	clean_input(boot_file_names[*file_num]);			
933	remove_comments(boot_file_names[*file_num]);			
934	if (SUCCESS == present(boot_file_names[*file_num], match_string)) {			
935	boot_files[*file_num] = boot_file_names[*file_num];			
936	LM_CHECK(lm_resolve_filename(lm_path, boot_files[*file_num],			
937	boot_files[*file_num], (int) max_str, (int) 0));			
938	(*file_num)++;			
939	}			
940				
941				
942	if (0 != *file_num) {			
943	(void) fclose(boot_list_file);			
944	return (SUCCESS);			
945	}			
946				
947	rewind(boot_list_file);			
948				
949	/* there were no entries found with a name on it */			
950				
951	while (NULL != fgets(boot_file_names[*file_num], max_str, boot_list_file)) {			
952	clean_input(boot_file_names[*file_num]);			
953	remove_comments(boot_file_names[*file_num]);			
954	if (SUCCESS == present(boot_file_names[*file_num], code_type)) {			
955	boot_files[*file_num] = boot_file_names[*file_num];			
956	LM_CHECK(lm_resolve_filename(lm_path, boot_files[*file_num],			
957	boot_files[*file_num], (int) max_str, (int) 0));			
958	(*file_num)++;			
959	}			
960				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 9/43
LINE #	SOURCE TEXT			
961	(void) fclose(boot_list_file);			
962	if (0 != *file_name) return(SUCCESS);			
963	if (0 != strcmp(code_type, "boot_diags"))			
964	(void) fprintf(stderr, "%s: no \"%s\" files found in \"%s\" for modeler \"%s\" \n",			
965	error_prefix, code_type, boot_list, modeler_name);			
966	else (void) fprintf(stderr, "%s: WARNING: boot diagnostics NOT run. \n", error_prefix);			
967	return(FAILURE);			
968	}			
969	present (string1, string2)			
970	char *string1, *string2;			
971	/* returns success if string2 is in string1 */			
972	/* oh, by the way, it changes string1 so that */			
973	/* it is truncated from the end of string2 */			
974	{			
975	char t1(max_str);			
976	char t2(max_str);			
977	(void) strcpy(t1, string1, max_str);			
978	(void) strcpy(t2, string2, max_str);			
979	clean_input(t1);			
980	clean_input(t2);			
981	(void) strcat(t2, ".");			
982	if (0 == strcmp(t1, t2, strlen(t2))) {			
983	if (0 == strchr(t1 + strlen(t2), '.')) {			
984	(void) strcpy(string1, t1 + strlen(t2));			
985	clean_input(string1);			
986	return(SUCCESS);			
987	}			
988	return(FAILURE);			
989	}			
990	}			
991	remove_comments (s)			
992	char *s;			
993	{			
994	char line(max_str);			
995	char *start_of_comment;			
996	(void) strcpy (line, s);			
997	if (0 != (start_of_comment = strchr(line, '#'))) {			
998	start_of_comment = (char) 0;			
999	}			
1000	}			
1001	void			
1002	lm_get_input (prompt, reply, len)			
1003	char *prompt;			
1004	char *reply;			
1005	short len;			
1006	{			
1007	(void) fprintf(stderr, "%s", prompt);			
1008	(void) fflush(stderr);			
1009	if (!fgets(reply, len, stdin) == 0) u_exit();			
1010	reply[strlen(reply) - 1] = 0;			
1011	clean_input (reply);			
1012	}			
1013	clear_counters()			
1014	{			
1015	RUNTIME_STAT_STRUCT counters;			
1016	char modeler_name(max_str);			
1017	long status;			
1018	status = get_parameter(PR_MODELER_NAME,			
1019	modeler_name, sizeof(modeler_name));			
1020	if (status != SUCCESS) return(FAILURE);			
1021	if (FAILURE == check_password(modeler_name))			
1022	return(FAILURE);			
1023	counters.dab_insertion_count = 0;			
1024	counters.pattern_count_mal = 0;			
1025	counters.pattern_count_lal = 0;			
1026	counters.longest_pattern_seq = 0;			
1027	counters.definition_count = 0;			
1028	counters.instance_count = 0;			
1029	counters.fault_count = 0;			
1030	LM_CHECK(lm_write(modeler_name, (u_long) LM_RAM_MEMORY, (u_long) 0, (u_long) RUNTIME_STAT,			
1031	(u_long) sizeof(RUNTIME_STAT, (char *) &counters, &status));			
1032	vprintf("Counters on modeler \"%s\" cleared\n", modeler_name);			
1033	return(SUCCESS);			
1034	}			
1035	abort_user ()			
1036	{			
1037	char modeler_name(max_str), user_number(max_str);			
1038	char reply(max_str);			
1039	long user_no, status, loop;			
1040	char *killed_user, *killed_host;			
1041	char *list_of_users, number_of_users, me;			
1042	char *list_of_user_names, *list_of_host_names;			
1043	status = get_parameter(PR_MODELER_NAME, modeler_name, sizeof(modeler_name));			
1044	if (status != SUCCESS) return(FAILURE);			
1045	LM_CHECK (lm_select_modeler (modeler_name));			
1046	if (FAILURE == check_password(modeler_name))			
1047	return(FAILURE);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 10/44
LINE #	SOURCE TEXT			
1081	status = get_parameter (PR_USER_NO, user_number, sizeof(user_number));			
1082	if (status != SUCCESS) return(FAILURE);			
1083				
1084	user_no = atoi(user_number);			
1085	lm_check (lm_inquire_user_list (number_of_users, list_of_users,			
1086	list_of_host_names, list_of_user_names));			
1087				
1088	for (loop = 0; loop < number_of_users; loop++,			
1089	list_of_users++, list_of_user_names++, list_of_host_names++) {			
1090	if ("list_of_users == user_no") {			
1091	killed_user = "list_of_user_names,			
1092	killed_host = "list_of_host_names,			
1093	break;			
1094	}			
1095				
1096				
1097				
1098	/* check to see that this is not me. */			
1099	/* you cannot delete yourself */			
1100				
1101	lm_check (lm_inquire_modeler (LM_WHICH_USER_AM_I, me));			
1102	if (user_no == me) {			
1103	(void) fprintf (stderr, "%s: Can only abort other users\n", error_prefix);			
1104	return (FAILURE);			
1105				
1106				
1107	vprintf ("Aborting user \"%s\" on \"%s\" \n", killed_user, killed_host);			
1108	lm_get_input ("Do you wish to continue? (Y/N) ", reply, sizeof(reply));			
1109	while (FAILURE == yes_or_no (reply)) {			
1110	lm_get_input ("Do you wish to continue? (Y/N) ", reply, sizeof(reply));			
1111				
1112				
1113	if (('n' == reply[0]) ('N' == reply[0])) {			
1114	vprintf ("%s",			
1115	return(FAILURE);			
1116				
1117				
1118	lm_check (lm_abort_user (user_no));			
1119				
1120	vprintf ("User number %d aborted on modeler \"%s\" (user \"%s\" on \"%s\" \n",			
1121	user_no, modeler_name, killed_user, killed_host);			
1122				
1123	return(SUCCESS);			
1124				
1125				
1126				
1127	install ()			
1128	{			
1129	char modeler_name[max_str], host_name[max_str],			
1130	char boot_type_str[max_str], port_string[max_str],			
1131	int port_number;			
1132	int status;			
1133	int boot_type;			
1134	BOOT_STRUCT boot_struct;			
1135				
1136	port_number = 1234;			
1137				
1138				
1139	status = get_parameter (PR_MODELER_NAME,			
1140	modeler_name, sizeof(modeler_name));			
1141	if (status != SUCCESS) return(FAILURE);			
1142				
1143	if (FAILURE == check_password(modeler_name))			
1144	return(FAILURE);			
1145				
1146	status = get_parameter (PR_AUTOROOT,			
1147	boot_type_str, sizeof(boot_type_str));			
1148	if (SUCCESS != status) return(FAILURE);			
1149				
1150	if (0 == strcmp (boot_type_str, "yes", strlen(boot_type_str)))			
1151	boot_type = AUTOROOT;			
1152	else boot_type = MANUALBOOT;			
1153				
1154	if (0 == strlen(boot_type_str)) boot_type = AUTOROOT;			
1155				
1156	if (AUTOROOT == boot_type) {			
1157	(void) strcpy (boot_name, host_name);			
1158				
1159	if (FALSE == command_mode)			
1160	if (FAILURE == set_default (PR_HOST_NAME, host_name))			
1161	return(FAILURE);			
1162				
1163	status = get_parameter (PR_HOST_NAME,			
1164	host_name, sizeof(host_name));			
1165	if (status != SUCCESS) return(FAILURE);			
1166				
1167	if ((Pe_mode == TRUE) (command_mode == TRUE)) {			
1168	status = get_parameter (PR_INET_PORT_NO,			
1169	port_string, sizeof(port_string));			
1170	if (status != SUCCESS) return(FAILURE);			
1171				
1172	port_number = atoi(port_string);			
1173				
1174				
1175	}			
1176	status = lm_create_boot_file (.ot_struct, modeler_name,			
1177	boot_type, boot_name, port_number);			
1178	if (status != SUCCESS) return(FAILURE);			
1179				
1180	vprintf ("Installed boot procedure for modeler \"%s\" \n", modeler_name);			
1181				
1182	return(SUCCESS);			
1183				
1184				
1185				
1186	suffix(result_file, base_file, suffix)			
1187	register char *result_file;			
1188	register char *base_file;			
1189	register char *suffix;			
1190				
1191	{			
1192	register char *s;			
1193				
1194	#ifdef VMS			
1195	if ((s = strrchr(base_file, '.')) != 0) {			
1196	(void) strcpy(result_file, ++s);			
1197	}			
1198	#else			
1199	if ((s = strrchr(base_file, '/')) != 0) {			
1200	(void) strcpy(result_file, ++s);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 11/45
LINE #	SOURCE TEXT			
1201	}			
1202	(void) strcpy(result_file, base_file);			
1203	#endif			
1204	if ((s = strchr(result_file, '.')) != 0) {			
1205	"s = \"0\";			
1206	}			
1207	(void) strcat(result_file, suffix);			
1208	}			
1209	}			
1210	play_vectors ()			
1211	{			
1212	char model_file(max_str);			
1213	char pattern_file(max_str);			
1214	char output_file(max_str);			
1215	char log_file(max_str);			
1216	int status;			
1217	status = get_parameter(PR_MODEL_NAME,			
1218	model_file, sizeof(model_file));			
1219	if (status != SUCCESS) return(FAILURE);			
1220	LM_CHECK(lm_resolve_library_file("LM_LIB",			
1221	model_file, model_file, max_str));			
1222	suffix (pattern_file, model_file, ".TST");			
1223	if (FALSE == command_mode)			
1224	if (FAILURE == set_default (PR_PATTERN_FILE, pattern_file))			
1225	return(FAILURE);			
1226	status = get_parameter(PR_PATTERN_FILE,			
1227	pattern_file, sizeof(pattern_file));			
1228	if (status != SUCCESS) return(FAILURE);			
1229	LM_CHECK(lm_resolve_library_file("LM_LIB",			
1230	pattern_file, pattern_file, max_str));			
1231	status = get_parameter(PR_OUTPUT_FILE,			
1232	output_file, sizeof(output_file));			
1233	if (status != SUCCESS) return(FAILURE);			
1234	(void) strcpy (log_file, "");			
1235	if (Fe_mode) {			
1236	status = get_parameter(PR_LOG_FILE,			
1237	log_file, sizeof(log_file));			
1238	if (status != SUCCESS) return(FAILURE);			
1239	}			
1240	vprintf("Playing vectors from file \"%s\\n", pattern_file);			
1241	(void) bobs_pattern_player (model_file,			
1242	pattern_file, output_file, log_file, "");			
1243	vprintf("Vector play completed\\n");			
1244	return(SUCCESS);			
1245	}			
1246	bobs_pattern_player (model_file_name, test_patterns, output_file, log_file, timing_file)			
1247	{			
1248	char *modname;			
1249	char *devname;			
1250	u_long nsets;			
1251	u_long nsets;			
1252	u_long ntimes;			
1253	u_long nmaps;			
1254	u_long ndiscs;			
1255	long definition;			
1256	long instance;			
1257	long real_time;			
1258	long done_time;			
1259	short status;			
1260	LM_CHECK(lm_set_simulation_tick (1, LM_FICOSECONDS));			
1261	LM_CHECK(lm_set_simulation_time ((unsigned int) 0, (unsigned int) 0));			
1262	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_ZERO, LM_LOGIC_ZERO));			
1263	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_SOFT_ZERO, LM_LOGIC_SOFT_ZERO));			
1264	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_ONE, LM_LOGIC_ONE));			
1265	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_SOFT_ONE, LM_LOGIC_SOFT_ONE));			
1266	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_FLOAT, LM_LOGIC_FLOAT));			
1267	LM_CHECK(lm_set_logic_level_map (LM_LOGIC_UNKNOWN, LM_LOGIC_UNKNOWN));			
1268	LM_CHECK(lm_create_definition_f (model_file_name, &definition, &modname, &devname));			
1269	status = lm_check(lm_create_instance (definition, &instance, devname));			
1270	(void) printf ("Playing to device \"%s\" on modeler \"%s\"\\n", devname, modname);			
1271	if (LM_SUCCESS == status) {			
1272	if (strlen(output_file) == 0)			
1273	output_file = LM_NO_FILE;			
1274	status = lm_check(lm_set_instance(instance));			
1275	if (LM_SUCCESS == status) {			
1276	if (strlen(log_file)>0)			
1277	(void) lm_check(lm_log_test_vectors(instance, LM_LOGGING_ON, log_file));			
1278	if (strlen(timing_file)>0)			
1279	(void) lm_check(lm_timing_measurement(instance, LM_TIMING_ON, timing_file));			
1280	real_time = (long) time((long *)0);			
1281	status = lm_check(lm_play_test_vectors (instance, test_patterns,			
1282	output_file, 0, &nsets,			
1283	&nsets, &ntimes, &nmaps, &ndiscs));			
1284	done_time = (long) time((long *)0);			
1285	}			
1286	}			
1287	if (LM_SUCCESS == status) {			
1288	vprintf("\\n");			
1289	vprintf(" Device: %s (%d signal pins) \\n", devname, nmaps);			
1290	vprintf(" Test Vectors: %d \\n", ntimes);			
1291	vprintf(" Pins set: %d \\n", nsets);			
1292	vprintf(" Pin Changes: %d \\n", nsets);			
1293	vprintf(" Discrepancies: %d \\n", ndiscs);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 12/46
LINE #	SOURCE TEXT			
1321	vprintf("Elapsed Time: %d seconds\n", (done_time - real_time));			
1322	vprintf("\n");			
1323	}			
1324	/* Clean up */			
1325				
1326				
1327	if (strlen(log_file)>0)			
1328	(void) lm_check(lm_log_test_vectors(instance, LM_LOGGING_OFF, log_file));			
1329				
1330	if (strlen(timing_file)>0)			
1331	(void) lm_check(lm_timing_measurement(instance, LM_TIMING_OFF, timing_file));			
1332				
1333	(void) lm_check(lm_release_instance(instance));			
1334	(void) lm_check(lm_release_definition(definition));			
1335				
1336	return(status);			
1337	}			
1338				
1339	create_timing_file ()			
1340	{			
1341	char input_file_list[max_str];			
1342	char output_file_name[max_str];			
1343	char bus_mode[max_str];			
1344	char next_file;			
1345	char *file_list;			
1346	char *file_pointer = NULL;			
1347	static FILE			
1348	bus_mode_flag;			
1349	int status;			
1350	char *delimiter;			
1351	char delimiter_char;			
1352				
1353	status = get_parameter(PR_TIM_FILE,			
1354	input_file_list, sizeof(input_file_list));			
1355	if (status != SUCCESS) return(FAILURE);			
1356				
1357	status = get_parameter(PR_TMC_FILE,			
1358	output_file_name, sizeof(output_file_name));			
1359	if (status != SUCCESS) return(FAILURE);			
1360				
1361	status = get_parameter(PR_BUS_MODE,			
1362	bus_mode, sizeof(bus_mode));			
1363	if (status != SUCCESS) return(FAILURE);			
1364				
1365	bus_mode_flag =			
1366	(('b' == bus_mode[0]) ('B' == bus_mode[0]))? FALSE: TRUE;			
1367				
1368	delimiter = strchr(input_file_list, ",");			
1369	if (delimiter != NULL) {			
1370	delimiter_char = *delimiter;			
1371	} else {			
1372	delimiter_char = ',';			
1373	}			
1374				
1375	file_list = input_file_list;			
1376	do {			
1377	next_file = strchr(file_list, delimiter_char);			
1378	if (next_file != NULL) {			
1379	*next_file = '\0'; next_file = NULL;			
1380	next_file++;			
1381	}			
1382	close(input_file_list);			
1383	if (file_pointer != NULL) (void) fclose(file_pointer);			
1384				
1385	file_pointer = fopen(file_list, "r");			
1386	if (file_pointer == NULL) {			
1387	(void) fprintf(stderr, "%s: unable to open file %s for reading\n",			
1388	error_prefix, file_list);			
1389	return(FAILURE);			
1390	}			
1391				
1392	vprintf("Processing file %s\n", file_list);			
1393	if (FAILURE == lm_filter_timing(file_pointer)) {			
1394	(void) fclose(file_pointer);			
1395	return(FAILURE);			
1396	}			
1397				
1398	file_list = next_file;			
1399	(void) fclose(file_pointer);			
1400				
1401	} while (next_file != NULL);			
1402				
1403	if (file_pointer != NULL) (void) fclose(file_pointer);			
1404				
1405	file_pointer = fopen(output_file_name, "w");			
1406	if (file_pointer == NULL) {			
1407	(void) fprintf(stderr, "%s: unable to open file %s for writing\n",			
1408	error_prefix, output_file_name);			
1409	return(FAILURE);			
1410	}			
1411				
1412	vprintf("Creating timing file %s\n", output_file_name);			
1413	if (FAILURE == lm_done_with_filter_timing(bus_mode_flag, file_pointer)) {			
1414	(void) fclose(file_pointer);			
1415	return(FAILURE);			
1416	}			
1417				
1418	vprintf("Timing file created\n");			
1419	(void) fclose(file_pointer);			
1420	return(SUCCESS);			
1421				
1422				
1423				
1424				
1425				
1426	locate_model_file ()			
1427	{			
1428	char filename[max_str];			
1429	char full_filename[max_str];			
1430	long status;			
1431				
1432				
1433	status = get_parameter(PR_FILE_NAME, filename, sizeof(filename));			
1434	if (status != SUCCESS) return(FAILURE);			
1435				
1436	status = lm_resolve_filename("LM_DIR", filename, full_filename, max_str, 0);			
1437	if (status != LM_SUCCESS) {			
1438	LM_CHECK(lm_resolve_library_file("LM_LIB", filename, full_filename, max_str));			
1439				
1440	}			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
lm/main.c

DATE 5/23/89
TIME 1:20:36 pm

PAGE #
13/47

```

1441
1442     printf ("%s \n", full_filename);
1443
1444     return(SUCCESS);
1445 }
1446
1447
1448
1449 measure_timing ()
1450 {
1451     char    model_file[max_str];
1452     char    pattern_file[max_str];
1453     char    output_file[max_str];
1454     char    timing_file[max_str];
1455     char    log_file[max_str];
1456     int     status;
1457
1458     status = get_parameter(PR_MODEL_NAME,
1459                          model_file, sizeof(model_file));
1460     if (status != SUCCESS) return(FAILURE);
1461     LM_CHECK(lm_resolve_library_file("LM_LIB", model_file, model_file, max_str));
1462     suffix (pattern_file, model_file, ".TST");
1463     if (FALSE == set_default (PR_PATTERN_FILE, pattern_file))
1464         return(FAILURE);
1465
1466     status = get_parameter(PR_PATTERN_FILE,
1467                          pattern_file, sizeof(pattern_file));
1468     if (status != SUCCESS) return(FAILURE);
1469     LM_CHECK(lm_resolve_library_file("LM_LIB", pattern_file, pattern_file, max_str));
1470     suffix (timing_file, model_file, ".TIM");
1471     if (FALSE == set_default (PR_TIMING_FILE, timing_file))
1472         return(FAILURE);
1473
1474     status = get_parameter(PR_TIMING_FILE,
1475                          timing_file, sizeof(timing_file));
1476     if (status != SUCCESS) return(FAILURE);
1477
1478     status = get_parameter(PR_OUTPUT_FILE,
1479                          output_file, sizeof(output_file));
1480     if (status != SUCCESS) return(FAILURE);
1481
1482     status = get_parameter(PR_LOG_FILE,
1483                          log_file, sizeof(log_file));
1484     if (status != SUCCESS) return(FAILURE);
1485
1486     (void) strcpy (log_file, "");
1487     if (Pr_mode) {
1488         status = get_parameter(PR_LOG_FILE,
1489                              log_file, sizeof(log_file));
1490         if (status != SUCCESS) return(FAILURE);
1491     }
1492
1493     vprintf("Playing vectors from file \"%s\" \n", pattern_file);
1494     (void) hoba_pattern_player (model_file, pattern_file, output_file, log_file, timing_file);
1495     vprintf("Vector play completed.\n");
1496
1497     return(SUCCESS);
1498 }
1499
1500
1501 set_baudrate ()
1502 {
1503     char    port[max_str];
1504     char    rate[max_str];
1505     char    modeler_name[max_str];
1506     long    status;
1507     char    real_rate;
1508
1509     status = get_parameter(PR_MODELER_NAME,
1510                          modeler_name, sizeof(modeler_name));
1511     if (status != SUCCESS) return(FAILURE);
1512
1513     status = get_parameter(PR_BAUD_RATE,
1514                          rate, sizeof(rate));
1515     if (status != SUCCESS) return(FAILURE);
1516
1517     status = get_parameter(PR_PORT,
1518                          port, sizeof(port));
1519     if (status != SUCCESS) return(FAILURE);
1520
1521     if (FAILURE == check_password(modeler_name))
1522         return(FAILURE);
1523
1524     real_rate = BAUD_96_12_24;
1525     if (0 == strcmp("300", rate)) real_rate = BAUD_300;
1526     if (0 == strcmp("1200", rate)) real_rate = BAUD_1200;
1527     if (0 == strcmp("2400", rate)) real_rate = BAUD_2400;
1528     if (0 == strcmp("4800", rate)) real_rate = BAUD_4800;
1529     if (0 == strcmp("9600", rate)) real_rate = BAUD_9600;
1530     if (0 == strcmp("2400+", rate)) real_rate = BAUD_24_96_12;
1531     if (0 == strcmp("9600+", rate)) real_rate = BAUD_96_12_24;
1532
1533     if (0 == strcmp("modem", port, strlen(port))) {
1534
1535         LM_CHECK(lm_write (modeler_name, LM_NVRAM_MEMORY, (u_long) 0,
1536                          (u_long) MODEM_BAUD, (u_long) sizeof(MODEM_BAUD,
1537                          (char *) &real_rate, &status));
1538
1539         switch (real_rate) {
1540             case (BAUD_24_96_12) : real_rate = BAUD_2400; break;
1541             case (BAUD_96_12_24) : real_rate = BAUD_9600; break;
1542         }
1543
1544         LM_CHECK(lm_write (modeler_name, LM_DUART_MODEM, (u_long) 0,
1545                          (u_long) 0, (u_long) 1,
1546                          (char *) &real_rate, &status));
1547     } else {
1548
1549         LM_CHECK(lm_write (modeler_name, (u_long) LM_NVRAM_MEMORY,
1550                          (u_long) 0, (u_long) CONSOLE_BAUD,
1551                          (u_long) sizeof(CONSOLE_BAUD,
1552                          (char *) &real_rate, &status));
1553
1554         switch (real_rate) {
1555             case (BAUD_24_96_12) : real_rate = BAUD_2400; break;
1556             case (BAUD_96_12_24) : real_rate = BAUD_9600; break;
1557         }
1558     }
1559 }
1560

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 14/48
LINE #	SOURCE TEXT			
1561	LM_CHECK(lm_write(modeler_name, (u_long) LM_DUART_CONSOLE,			
1562	(u_long) 0, (u_long) 0, (u_long) 1,			
1563	(char *) ’_Fats, &status));			
1564	}			
1565	return(SUCCESS);			
1566	}			
1567	}			
1568	}			
1569	}			
1570	label()			
1571	{			
1572	int status;			
1573	char lane[20], slot[20];			
1574	device_name(max_str), manufacturer(max_str),			
1575	modeler_name(max_str), dab_type(max_str), revision(max_str);			
1576	char maker(max_str), maker_rev(max_str);			
1577	long lane_number, slot_number;			
1578	int lanes_high;			
1579	int slots_wide;			
1580	segment_number;			
1581	long dab_number, device_number;			
1582	char height(max_str), width(max_str);			
1583	char seg_str(max_str);			
1584	}			
1585	segment_number = 0;			
1586	}			
1587	status = get_parameter(PR_MODELER_NAME,			
1588	modeler_name, sizeof(modeler_name));			
1589	if (status != SUCCESS) return(FAILURE);			
1590	LM_CHECK(lm_select_modeler(modeler_name));			
1591	do {			
1592	status = get_parameter(PR_LANE,			
1593	lane, sizeof(lane));			
1594	if (status != SUCCESS) return(FAILURE);			
1595	status = get_parameter(PR_SLOT,			
1596	slot, sizeof(slot));			
1597	if (status != SUCCESS) return(FAILURE);			
1598	if (isupper(lane[0])) lane[0] = tolower(lane[0]);			
1599	lane_number = lane[0] - 'a';			
1600	slot_number = atoi(slot);			
1601	} while (((command_mode) & ((Fe_mode) & ((FAILURE ==			
1602	check_for_dab (lane_number, slot_number, &device_number, &dab_number))));			
1603	status = get_parameter(PR_DEVICE_NAME,			
1604	device_name, sizeof(device_name));			
1605	if (status != SUCCESS) return(FAILURE);			
1606	status = get_parameter(PR_DAB MAKER,			
1607	maker, sizeof(maker));			
1608	if (status != SUCCESS) return(FAILURE);			
1609	status = get_parameter(PR_DAB MAKER REV,			
1610	maker_rev, sizeof(maker_rev));			
1611	if (status != SUCCESS) return(FAILURE);			
1612	if ((Fe_mode == TRUE) ((command_mode == TRUE) & ((FAILURE ==			
1613	check_for_dab (lane_number, slot_number, &device_number, &dab_number)))) {			
1614	status = get_parameter(PR_SEGMENT,			
1615	seg_str, sizeof(seg_str));			
1616	if (status != SUCCESS) return(FAILURE);			
1617	status = get_parameter(PR_MANUFACTURER,			
1618	manufacturer, sizeof(manufacturer));			
1619	if (status != SUCCESS) return(FAILURE);			
1620	status = get_parameter(PR_REVISION,			
1621	revision, sizeof(revision));			
1622	if (status != SUCCESS) return(FAILURE);			
1623	status = get_parameter(PR_DAB TYPE,			
1624	dab_type, sizeof(dab_type));			
1625	if (status != SUCCESS) return(FAILURE);			
1626	status = get_parameter(PR_LANES HIGH,			
1627	height, sizeof(height));			
1628	if (status != SUCCESS) return(FAILURE);			
1629	status = get_parameter(PR_SLOTS WIDE,			
1630	width, sizeof(width));			
1631	if (status != SUCCESS) return(FAILURE);			
1632	slots_wide = atoi (width);			
1633	lanes_high = atoi (height);			
1634	}			
1635	} else if ((command_mode == TRUE) & ((SUCCESS ==			
1636	check_for_dab (lane_number, slot_number, &device_number, &dab_number)))){			
1637	long dab, device_number;			
1638	long number_of_locations, *lanes, *slots;			
1639	long max_slot, min_slot, max_lane, min_lane;			
1640	pos;			
1641	char *ret_string;			
1642	if (FAILURE == check_for_dab (lane_number, slot_number, &device_number, &dab))			
1643	return(FAILURE);			
1644	/* inquire device to determine size and upper left slot */			
1645	LM_CHECK (lm_inquire_dab_location (device_number, dab,			
1646	number_of_locations, &lanes, &slots));			
1647	min_lane = lanes[0]; min_slot = slots[0];			
1648	max_lane = lanes[0]; max_slot = slots[0];			
1649	for (pos = 0; pos < number_of_locations; pos++) {			
1650	if (lanes[pos] < min_lane) min_lane = lanes[pos];			
1651	if (slots[pos] < min_slot) min_slot = slots[pos];			
1652	if (lanes[pos] > max_lane) max_lane = lanes[pos];			
1653	if (slots[pos] > max_slot) max_slot = slots[pos];			
1654	}			
1655	lane_number = min_lane;			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 15/49
LINE #	SOURCE TEXT			
1681	slot_number = min_slot;			
1682				
1683	lanes_high = 1 + (max_lane - min_lane);			
1684	slots_wide = 1 + (max_slot - min_slot);			
1685				
1686	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_TYPE, &ret_string));			
1687	(void) strcpy(dab_type, ret_string);			
1688				
1689	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_REVISION, &ret_string));			
1690	(void) strcpy(revision, ret_string);			
1691				
1692	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_MANUFACTURER, &ret_string));			
1693	(void) strcpy(manufacturer, ret_string);			
1694				
1695	if (is_param_defined(PR_SEGMENT)) {			
1696	status = get_parameter(PR_SEGMENT, seg_str, sizeof(seg_str));			
1697	if (status != SUCCESS) return(FAILURE);			
1698	}			
1699				
1700	if (is_param_defined(PR_MANUFACTURER)) {			
1701	status = get_parameter(PR_MANUFACTURER, manufacturer, sizeof(manufacturer));			
1702	if (status != SUCCESS) return(FAILURE);			
1703	}			
1704				
1705	if (is_param_defined(PR_REVISION)) {			
1706	status = get_parameter(PR_REVISION, revision, sizeof(revision));			
1707	if (status != SUCCESS) return(FAILURE);			
1708	}			
1709				
1710	if (is_param_defined(PR_DAB_TYPE)) {			
1711	status = get_parameter(PR_DAB_TYPE, dab_type, sizeof(dab_type));			
1712	if (status != SUCCESS) return(FAILURE);			
1713	}			
1714				
1715	if (is_param_defined(PR_LANES_HIGH)) {			
1716	status = get_parameter(PR_LANES_HIGH, height, sizeof(height));			
1717	if (status != SUCCESS) return(FAILURE);			
1718	lanes_high = atoi(height);			
1719	}			
1720				
1721	if (is_param_defined(PR_SLOTS_WIDE)) {			
1722	status = get_parameter(PR_SLOTS_WIDE, width, sizeof(width));			
1723	if (status != SUCCESS) return(FAILURE);			
1724	slots_wide = atoi(width);			
1725	}			
1726				
1727				
1728	} else {			
1729				
1730	long dab, device_number;			
1731	long number_of_locations, *lanes, *slots;			
1732	long max_slot, min_slot, max_lane, min_lane;			
1733	long pos;			
1734	char *ret_string;			
1735				
1736	if (FAILURE == check_for_dab(lane_number, slot_number, &device_number, &dab))			
1737	return(FAILURE);			
1738				
1739	/* Inquire device to determine size and upper left slot */			
1740				
1741	LM_CHECK(lm_inquire_dab_location(device_number, dab,			
1742	number_of_locations, &lanes, &slots));			
1743				
1744	min_lane = lanes[0]; min_slot = slots[0];			
1745	max_lane = lanes[0]; max_slot = slots[0];			
1746				
1747	for (pos = 0; pos < number_of_locations; pos++) {			
1748	if (lanes[pos] < min_lane) min_lane = lanes[pos];			
1749	if (slots[pos] < min_slot) min_slot = slots[pos];			
1750	if (lanes[pos] > max_lane) max_lane = lanes[pos];			
1751	if (slots[pos] > max_slot) max_slot = slots[pos];			
1752	}			
1753				
1754	lane_number = min_lane;			
1755	slot_number = min_slot;			
1756				
1757	lanes_high = 1 + (max_lane - min_lane);			
1758	slots_wide = 1 + (max_slot - min_slot);			
1759				
1760	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_TYPE, &ret_string));			
1761	(void) strcpy(dab_type, ret_string);			
1762				
1763	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_REVISION, &ret_string));			
1764	(void) strcpy(revision, ret_string);			
1765				
1766	LM_CHECK(lm_inquire_dab(device_number, dab, LM_DAB_MANUFACTURER, &ret_string));			
1767	(void) strcpy(manufacturer, ret_string);			
1768				
1769				
1770				
1771	LM_CHECK(lm_label_dab(lane_number, slot_number, dab_type, device_name, maker, maker_rev,			
1772	manufacturer, revision, lanes_high, slots_wide, segment_number));			
1773				
1774	vprintf("\n");			
1775	vprintf("Labeled Device Adapter in lane %c, slot %d\n", 'A' + lane_number, slot_number);			
1776	vprintf("\n");			
1777				
1778	return(SUCCESS);			
1779				
1780				
1781				
1782				
1783	check_for_dab(lane, slot, device_number, dab_number)			
1784	long lane;			
1785	long slot;			
1786	long *device_number;			
1787	long *dab_number;			
1788				
1789	long number_of_devices, *list_of_devices;			
1790	long number_of_dabs, dab, device;			
1791	long number_of_locations, *lanes, *slots;			
1792	long pos;			
1793	char **list_of_device_names;			
1794				
1795	/* return success if there is a dab here */			
1796				
1797	LM_CHECK(lm_inquire_device_list(number_of_devices, &list_of_devices, &list_of_device_names))			
1798				
1799	for (device = 0; device < number_of_devices; device++, list_of_devices++, list_of_device_names++) {			
1800	LM_CHECK(lm_inquire_device(*list_of_devices, LM_NUMBER_OF_DABS, number_of_dabs));			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89	PAGE # 16/50
LINE #	SOURCE TEXT			
1801	for (dab = 0; dab < number_of_dabs; dab++) {			
1802	LM_CHECK(lm_inquire_dab_location(list_of_devices, dab, number_of_locations, lanes, slots));			
1803	for (pos = 0; pos < number_of_locations; pos++) {			
1804	if ((lanes[pos] == lane) && (slots[pos] == slot)) {			
1805	device_number = list_of_devices;			
1806	dab_number = dab;			
1807	return (SUCCESS);			
1808	}			
1809	}			
1810	}			
1811	}			
1812	(void) fprintf(stderr, "No device adapter installed in lane %d slot %d\n",			
1813	error_prefix, 'A' + lane, slot);			
1814	return (FAILURE);			
1815	}			
1816	}			
1817	}			
1818	}			
1819	}			
1820	check_shell_software ()			
1821	{			
1822	char model_name[max_str];			
1823	int status;			
1824	int errors, warnings;			
1825	char message;			
1826	int severity;			
1827	errors = 0;			
1828	warnings = 0;			
1829	status = get_parameter(PR_MODEL_NAME,			
1830	model_name, sizeof(model_name));			
1831	if (status != SUCCESS) return(FAILURE);			
1832	status = lm_check_shell_software_f(model_name);			
1833	if (LM_SUCCESS != status) {			
1834	do {			
1835	lm_get_message(&severity, &message);			
1836	if (severity == LM_ERROR) {			
1837	(void) fprintf(stderr,			
1838	"LM modeler error: %s\n", message);			
1839	errors ++;			
1840	}			
1841	if (severity == LM_WARNING) {			
1842	(void) fprintf(stderr,			
1843	"LM modeler warning: %s\n", message);			
1844	warnings ++;			
1845	}			
1846	} while (severity != LM_NO_MORE_MESSAGES);			
1847	}			
1848	vprintf("\nModel \"%s\" checked with ", model_name);			
1849	if (0 == errors)			
1850	vprintf("no errors ");			
1851	else if (1 == errors)			
1852	vprintf("1 error ");			
1853	else vprintf("%d errors ", errors);			
1854	vprintf("\n");			
1855	if (0 == warnings)			
1856	vprintf("no warnings ");			
1857	else if (1 == warnings)			
1858	vprintf("1 warning ");			
1859	else vprintf("%d warnings ", warnings);			
1860	vprintf("\n");			
1861	return(SUCCESS);			
1862	}			
1863	}			
1864	}			
1865	}			
1866	}			
1867	}			
1868	}			
1869	}			
1870	}			
1871	}			
1872	}			
1873	}			
1874	re_boot()			
1875	{			
1876	char modeler_name[max_str];			
1877	char wait_string[max_str], seconds_string[max_str];			
1878	long status;			
1879	int wait, seconds;			
1880	BOOT_STRUCT boot_save;			
1881	status = get_parameter(PR_MODELER_NAME,			
1882	modeler_name, sizeof(modeler_name));			
1883	if (status != SUCCESS) return(FAILURE);			
1884	if (FAILURE == check_password(modeler_name))			
1885	return(FAILURE);			
1886	if (SUCCESS != check_users(modeler_name))			
1887	return (FAILURE);			
1888	if (SUCCESS != check_users(modeler_name))			
1889	return (FAILURE);			
1890	if (number_of_users(modeler_name) > 1) {			
1891	if (get_parameter(PR_WAIT, wait_string,			
1892	sizeof(wait_string)) != SUCCESS) return(FAILURE);			
1893	if ((wait_string[0] == 'Y') (wait_string[0] == 'y'))			
1894	wait = LM_TRUE; else wait = LM_FALSE;			
1895	} else wait = LM_FALSE;			
1896	if (get_parameter(PR_MINUTES, seconds_string,			
1897	sizeof(seconds_string)) != SUCCESS) return(FAILURE);			
1898	seconds = 60 * (atoi (seconds_string));			
1899	vprintf("Rebooting modeler \"%s\" ", modeler_name);			
1900	if (seconds > 0)			
1901	vprintf("in %d minutes ", seconds/60);			
1902	if (wait)			
1903	vprintf("after all modeling sessions complete ");			
1904	vprintf("\n");			
1905	LM_CHECK(lm_select_modeler(modeler_name));			
1906	LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,			
1907	(u_long) 0, (u_long) 0, (u_long) sizeof(BOOT_STRUCT),			
1908	(char *) &boot_save, &status));			
1909	LM_CHECK(lm_reboot(seconds, wait));			
1910	if ((wait) (boot_save.autoboot != LM_FALSE)) {			
1911	vprintf ("Modeler \"%s\" will autoboot shortly \n", modeler_name);			
1912	return (SUCCESS);			
1913	}			
1914	}			
1915	}			
1916	}			
1917	}			
1918	}			
1919	}			
1920	}			

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM	DATE	PAGE #
	lm/main.c	5/23/89	17/51
		TIME	1:20:36 pm

LINE #	SOURCE TEXT
1921	1
1922	1
1923	/* Reboot the modeler with core modeler code */
1924	
1925	return (download_modeler_code (modeler_name, "modeler_code", BOOT_LIST));
1926	
1927	1
1928	
1929	
1930	
1931	boot ()
1932	{
1933	char modeler_name(max_str);
1934	char bootlist(max_str);
1935	char mode_string(max_str);
1936	u_short mode;
1937	int status;
1938	
1939	mode = TRUE;
1940	(void) strcpy (bootlist, BOOT_LIST);
1941	
1942	/* get parameters from the command line */
1943	
1944	status = get_parameter(PR_MODELER_NAME,
1945	modeler_name, sizeof(modeler_name));
1946	if (status != SUCCESS) return(FAILURE);
1947	
1948	if ((!mode == TRUE) (command_mode == TRUE)) {
1949	status = get_parameter(PR_BOOT_LIST,
1950	bootlist, sizeof(bootlist));
1951	if (status != SUCCESS) return(FAILURE);
1952	
1953	status = get_parameter(PR_MODE,
1954	mode_string, sizeof(mode_string));
1955	if (status != SUCCESS) return(FAILURE);
1956	
1957	if ((mode_string[0] == 'd') (mode_string[0] == 'D'))
1958	mode = FALSE;
1959	
1960	}
1961	
1962	if (!mode)
1963	return(download_modeler_code (modeler_name, "diagnostics", bootlist));
1964	
1965	if (SUCCESS == download_modeler_code (modeler_name, "boot_diags", bootlist)) {
1966	
1967	if (FAILURE == all_test(modeler_name)) {
1968	vprintf ("Modeler \"%s\" failed diagnostics\n", modeler_name);
1969	return(FAILURE);
1970	}
1971	
1972	if (FAILURE == download_modeler_code (modeler_name, "modeler_code", bootlist))
1973	return(FAILURE);
1974	
1975	return(SUCCESS);
1976	
1977	1
1978	
1979	
1980	
1981	log_boot(modeler_name)
1982	{
1983	char modeler_name;
1984	
1985	char filename(max_str);
1986	char modeler_state;
1987	int position;
1988	static FILE *stream = NULL;
1989	long clock;
1990	char *time_stamp_pointer;
1991	char time_stamp[26];
1992	long status;
1993	char log_filename(max_str);
1994	int new_file;
1995	
1996	modeler_state = -1;
1997	LM_CHECK (lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,
1998	(u_long) 0, (u_long) MODELER_STATE,
1999	(u_long) sizeof MODELER_STATE,
2000	(char *) &modeler_state, &status));
2001	
2002	switch (modeler_state) {
2003	case SHUTDOWN:
2004	vprintf("modeler \"%s\" was shutdown\n", modeler_name);
2005	(void) check_errors(modeler_name);
2006	break;
2007	case REBOOT:
2008	vprintf("modeler \"%s\" was rebooted\n", modeler_name);
2009	break;
2010	case BOOTING:
2011	vprintf("modeler \"%s\" is waiting to be booted\n",
2012	modeler_name);
2013	break;
2014	case BOOTING:
2015	vprintf("modeler \"%s\" is being booted\n", modeler_name);
2016	return (FAILURE);
2017	case BOOTED:
2018	vprintf("modeler \"%s\" is booted\n",
2019	modeler_name);
2020	return (FAILURE);
2021	case RUNNING_DIAGS:
2022	vprintf("modeler \"%s\" is running diagnostics\n",
2023	modeler_name);
2024	return (FAILURE);
2025	case MODELER_RUNNING:
2026	vprintf("modeler \"%s\" is running\n",
2027	modeler_name);
2028	return (FAILURE);
2029	default:
2030	vprintf("modeler \"%s\" was in an unknown state: %04x\n",
2031	modeler_name, modeler_state);
2032	break;
2033	}
2034	
2035	(void) strcpy (log_filename, modeler_name);
2036	(void) strcat (log_filename, ".log");
2037	
2038	LM_CHECK (lm_resolve_filename (lm_path,
2039	log_filename, filename, (int) max_str, (int) 2));
2040	

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 18/52
LINE #	SOURCE TEXT			
2041	new_file = access (filename, 0);			
2042				
2043	if (stream != NULL) (void) fclose(stream);			
2044	stream = fopen (filename, "a");			
2045	if (new_file) {			
2046	/* try to set the protection to -rw-rw-rw- */			
2047	(void) chmod(filename, 0666);			
2048	}			
2049	if (NULL == stream) {			
2050	(void) fprintf(stderr, "%s: unable to open modeler log file \"%s\".\n",			
2051	error_prefix, filename);			
2052	return(FAILURE);			
2053	}			
2054				
2055	clock = (long) time(&time_stamp_pointer);			
2056	time_stamp_pointer = ctime(&clock);			
2057	(void) strcpy (time_stamp, time_stamp_pointer);			
2058	if (SUCCESS == russ_index(time_stamp, "\n", &position))			
2059	time_stamp[position] = NULL;			
2060				
2061	(void) fprintf(stream, "%s Modeler booted by %s from %s\n",			
2062	time_stamp, User_name, Host_name);			
2063	(void) fclose(stream);			
2064				
2065	modeler_state = BOOTING;			
2066				
2067	LM_CHECK (lm_write (modeler_name, (u_long) LM_NVRAM_MEMORY,			
2068	(u_long) 0, (u_long) MODELER_STATE,			
2069	(u_long) sizeof MODELER_STATE,			
2070	(char *) &modeler_state, &status));			
2071				
2072	return (SUCCESS);			
2073				
2074	}			
2075				
2076				
2077				
2078	check_errors (modeler_name)			
2079	char *modeler_name;			
2080				
2081	{			
2082	char error_condition;			
2083	long status;			
2084	FILE *stream = NULL;			
2085	static char filename[max_str];			
2086	char log_filename[max_str];			
2087	int new_file;			
2088				
2089	(void) strcpy(log_filename, modeler_name);			
2090	(void) strcat(log_filename, ".log");			
2091				
2092	LM_CHECK (lm_resolve_filename(lm_path,			
2093	log_filename, filename, (int) max_str, (int) 2));			
2094				
2095	if (stream != NULL) (void) fclose(stream);			
2096	new_file = access (filename, 0);			
2097	stream = fopen (filename, "a");			
2098	if (new_file) {			
2099	/* try to set the protection to -rw-rw-rw- */			
2100	(void) chmod(filename, 0666);			
2101	}			
2102	if (NULL == stream) {			
2103	(void) fprintf(stderr, "%s: Unable to open file: \"%s\".\n",			
2104	error_prefix, filename);			
2105	return(FAILURE);			
2106	}			
2107				
2108	LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,			
2109	(u_long) 0, (u_long) MODELER_RESET,			
2110	(u_long) sizeof MODELER_RESET, (char *) &error_condition,			
2111	&status));			
2112				
2113	if ((error_condition != 0) &&			
2114	(error_condition != SHUTDOWN) &&			
2115	(error_condition != REMOTE_RESET))			
2116	{			
2117	(void) fprintf(stream,			
2118	".....\n",			
2119	(void) fprintf(stream,			
2120	"%s\n", "Fatal hardware error has been detected.\n");			
2121	(void) fprintf(stream,			
2122	"%s\n", "Run diagnostics or call Field Engineering.\n");			
2123	}			
2124				
2125	switch (error_condition) {			
2126				
2127	case 0:			
2128	case SHUTDOWN:			
2129	case REMOTE_RESET:			
2130	(void) fclose(stream);			
2131	return (SUCCESS);			
2132				
2133	case PARITY_ERROR:			
2134	(void) log_parity_error (modeler_name, stream);			
2135	break;			
2136				
2137	case BACKPLANE_ERROR:			
2138	(void) log_backplane_error (modeler_name, stream);			
2139	break;			
2140				
2141	case BUS_ERROR:			
2142	(void) log_bus_error (modeler_name, stream);			
2143	break;			
2144				
2145	case SUICIDE:			
2146	(void) fprintf(stream, "SUICIDE\n");			
2147	break;			
2148				
2149	case LANCE_ERROR:			
2150	(void) log_lance_error (modeler_name, stream);			
2151	break;			
2152				
2153	case BAD_NVRAM:			
2154	(void) log_nva_ram_error (stream);			
2155	break;			
2156				
2157	case OUTOFDATE_EPROMS:			
2158	(void) log_old_eproms (stream);			
2159	break;			
2160				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89	PAGE # 19/53
LINE #		SOURCE TEXT		
2161		case OUTOFDATE_HOST_UTILS:		
2162		(void) log_old_utils (stream);		
2163		break;		
2164				
2165		default:		
2166		(void) fprintf(stream, "An unknown error occured. ");		
2167		(void) fprintf(stream, "Error code: 0x%x\n",		
2168		error_condition);		
2169		break;		
2170				
2171				
2172				
2173		(void) fprintf(stream,		
2174	\n");		
2175				
2176		error_condition = 0; /* reset the error condition */		
2177				
2178		LM_CHECK(lm_write(modeler_name, (u_long) LM_NVRAM_MEMORY, (u_long) 0,		
2179		(u_long) MODELER_STATE, (u_long) SILEOF_MODELER_STATE,		
2180		(char *)error_condition, &status));		
2181				
2182		(void) fclose(stream);		
2183		return (SUCCESS);		
2184				
2185				
2186				
2187		log_parity_error (modeler_name, stream)		
2188		char *modeler_name;		
2189		FILE *stream;		
2190				
2191		{		
2192		cpu_par_err_reg parity_error;		
2193		long status;		
2194				
2195		LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,		
2196		(u_long) 0, (u_long) PARITY_ERR,		
2197		(u_long) SILEOF_PARITY_ERR, (char *) &parity_error, &status));		
2198				
2199		(void) fprintf(stream, "PARITY ERROR ");		
2200		(void) fprintf(stream, " 0x%x ", parity_error);		
2201				
2202		(void) fprintf(stream, " - Bus Controlled by: ");		
2203				
2204		if (!(parity_error.not_68020_master)) (void) fprintf(stream, "CPU ");		
2205		else if (!(parity_error.not_VME_master)) (void) fprintf(stream, "VME ");		
2206		else if (!(parity_error.not_LANCI_master)) (void) fprintf(stream, "LANCI ");		
2207		else (void) fprintf(stream, "unknown ");		
2208				
2209		(void) fprintf(stream, " - Byte: ");		
2210				
2211		if (!(parity_error.not_error_hi)) (void) fprintf(stream, "1 ");		
2212		if (!(parity_error.not_error_um)) (void) fprintf(stream, "2 ");		
2213		if (!(parity_error.not_error_lm)) (void) fprintf(stream, "3 ");		
2214		if (!(parity_error.not_error_lo)) (void) fprintf(stream, "4 ");		
2215				
2216		(void) fprintf(stream, " - Address: ");		
2217		(void) fprintf(stream, "0x%x\n", (parity_error.error_addr << 2));		
2218		return (SUCCESS);		
2219				
2220				
2221		#define NUMBER_OF_MAGIC_CHIPS 5		
2222				
2223		log_backplane_error (modeler_name, stream)		
2224		char *modeler_name;		
2225		FILE *stream;		
2226		{		
2227		/* Record backplane error in logfile */		
2228				
2229		LM_HARDWARE_ERROR hardware;		
2230		long status;		
2231		int lane, slot, pel, chip, bit;		
2232		u_long pel_control, data_valid;		
2233				
2234		(void) fprintf(stream, "BACKPLANE ERROR ON MODELER %s\n",		
2235		modeler_name);		
2236				
2237		LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,		
2238		(u_long) 0, (u_long) HARDWARE_ERROR,		
2239		(u_long) SILEOF_HARDWARE_ERROR, (char *) &hardware, &status));		
2240				
2241		if (hardware.error != 0)		
2242		(void) fprintf(stream, "FATAL HARDWARE ERROR DETECTED\n");		
2243		/* also return out of this routine ??? */		
2244				
2245		if (hardware.timing_error != 0) {		
2246		(void) fprintf(stream, "Timing Generator error:\n");		
2247		for (lane = 0; lane < NUMBER_OF_LANES; lane++) {		
2248		if (((1 & (hardware.lane_enable >> lane)) != 0) {		
2249		switch (lane) {		
2250		case (0) :		
2251		pel_control = hardware.lane_a_pel_control;		
2252		data_valid = hardware.lane_a_data_valid;		
2253		break;		
2254		case (1) :		
2255		pel_control = hardware.lane_b_pel_control;		
2256		data_valid = hardware.lane_b_data_valid;		
2257		break;		
2258		case (2) :		
2259		pel_control = hardware.lane_c_pel_control;		
2260		data_valid = hardware.lane_c_data_valid;		
2261		break;		
2262		case (3) :		
2263		pel_control = hardware.lane_d_pel_control;		
2264		data_valid = hardware.lane_d_data_valid;		
2265		break;		
2266		};		
2267		(void) fprintf(stream,		
2268		"lane %c enabled, PEL control %x, Data valid %d\n",		
2269		'A' + lane, pel_control, data_valid);		
2270				
2271		}		
2272				
2273				
2274		for (lane = 0; lane < NUMBER_OF_LANES; lane++) {		
2275		if (((hardware.lane_errors >> lane) & 1) == 0)		
2276		continue;		
2277				
2278		(void) fprintf(stream, "Lane %c errors:\n", 'A' + (char) lane);		
2279				
2280		if (((hardware.pac_lane_errors >> lane) & 1) != 0) {		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 21/55
LINE #	SOURCE TEXT			
2401	(void) fprintf(stream, "\t Fault during READ ");			
2402	else			
2403	(void) fprintf(stream, "\t Fault during WRITE ");			
2404	size = ((0x0010 & status_reg) >> 4);			
2405	if (size == 0) (void) fprintf(stream, "BYTE operation\n");			
2406	if (size == 1) (void) fprintf(stream, "WORD operation\n");			
2407	if (size == 2) (void) fprintf(stream, "LONGWORD operation\n");			
2408	return (SUCCESS);			
2409	}			
2410				
2411				
2412	log_lance_error (modeler_name, stream)			
2413	CHAR			
2414	modeler_name;			
2415	FILE			
2416	stream;			
2417	{			
2418	u_short lance_error;			
2419	long			
2420	status;			
2421	(void) fprintf(stream, "LANCE ERROR");			
2422				
2423	LM_CHECK(lm_read(modeler_name, (u_long) LM_NVRAM_MEMORY,			
2424	(u_long) 0, (u_long) LANCE_CSR_REG,			
2425	(u_long) sizeof LANCE_CSR_REG, (char *) &lance_error, &status);			
2426	(void) fprintf(stream, " 0x%x\n", lance_error);			
2427				
2428	if (1 & (lance_error >> 14))			
2429	(void) fprintf(stream, "\t Transmitter timeout error\n");			
2430	if (1 & (lance_error >> 13))			
2431	(void) fprintf(stream, "\t Collision error\n");			
2432	if (1 & (lance_error >> 12))			
2433	(void) fprintf(stream,			
2434	"\t Missed packet. No space in receive buffers\n");			
2435	if (1 & (lance_error >> 11))			
2436	(void) fprintf(stream, "\t Memory error detected by LANCE\n");			
2437	if (1 & (lance_error >> 5))			
2438	(void) fprintf(stream, "\t LANCE transmitter turned off\n");			
2439	return (SUCCESS);			
2440	}			
2441				
2442				
2443				
2444	log_nvs_ram_error (stream)			
2445	FILE			
2446	stream;			
2447	{			
2448	(void) fprintf(stream,			
2449	"NVS RAM has been corrupted. Modeler must be re-installed\n");			
2450				
2451	}			
2452				
2453	log_old_oproms (stream)			
2454	FILE			
2455	stream;			
2456	{			
2457	(void) fprintf(stream,			
2458	"CPU EPROMs are not the current version, they must be replaced. \n");			
2459				
2460	}			
2461	log_old_utils (stream)			
2462	FILE			
2463	stream;			
2464	{			
2465	(void) fprintf(stream,			
2466	"An old version of the install procedure was used to install\n");			
2467	(void) fprintf(stream,			
2468	"this modeler. The modeler must be re-installed with a newer version \n");			
2469	(void) fprintf(stream,			
2470	"of the installation program \n");			
2471				
2472	}			
2473				
2474				
2475				
2476				
2477	shut_down ()			
2478	{			
2479	char			
2480	modeler_name[MAX_STR];			
2481	char			
2482	wait_string[MAX_STR];			
2483	int			
2484	wait, seconds, status;			
2485	char			
2486	reply[100];			
2487	char			
2488	state;			
2489				
2490	status = get_parameter(PR_MODELER_NAME,			
2491	modeler_name, sizeof(modeler_name));			
2492	if (status != SUCCESS) return(FAILURE);			
2493	if (FAILURE == check_password(modeler_name))			
2494	return(FAILURE);			
2495	if (SUCCESS != check_users(modeler_name))			
2496	return (FAILURE);			
2497	if (number_of_users(modeler_name)>1) {			
2498	status = get_parameter(PR_WAIT,			
2499	wait_string, sizeof(wait_string));			
2500	if (status != SUCCESS) return(FAILURE);			
2501	if ((wait_string[0] == 'Y') (wait_string[0] == 'y'))			
2502	wait = LM_TRUE; else wait = LM_FALSE;			
2503	} else wait = LM_FALSE;			
2504	status = get_parameter(PR_MINUTES,			
2505	seconds_string, sizeof(seconds_string));			
2506	if (status != SUCCESS) return(FAILURE);			
2507	seconds = 60 * (atoi (seconds_string));			
2508	(void) lm_check(lm_select_modeler(modeler_name));			
2509	status = lm_shutdown(seconds, wait);			
2510				
2511				
2512	if (LM_ERROR == status) {			
2513	lm_get_input ("Do you wish to force modeler shutdown? (Y/N) ", reply, sizeof(reply));			
2514	while (FAILURE == yes_or_no (reply)) {			
2515	lm_get_input ("Do you wish to force modeler shutdown? (Y/N) ", reply, sizeof(reply));			
2516	}			
2517	if (('Y' == reply[0]) ('y' == reply[0])) {			
2518	LM_CHECK(lm_remote_reset(modeler_name, &state));			
2519	fprintf("Modeler \"%s\" shutdown\n", modeler_name);			
2520	return(SUCCESS);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89	PAGE # 22/56
LINE #		SOURCE TEXT		
2521		}		
2522		else return(FAILURE);		
2523		}		
2524		(void) lm_check(status);		
2525		if ((seconds == 0) && (!wait))		
2526		vprintf("Modeler \"%s\" shutdown\n", modeler_name);		
2527		else {		
2528		vprintf("Shutting down modeler \"%s\" ", modeler_name);		
2529		if (seconds > 0)		
2530		vprintf("in %d minutes ", seconds/60);		
2531		if (LM_TIME == wait)		
2532		vprintf("after all modeling sessions complete ");		
2533		vprintf("\n");		
2534		}		
2535		return(SUCCESS);		
2536		}		
2537		}		
2538		}		
2539		}		
2540		}		
2541		}		
2542		}		
2543		}		
2544		lm_create_boot_file (boot_struct, modeler_name, boot_type, host_name, port_number)		
2545		{		
2546		BOOT_STRUCT boot_struct;		
2547		char *modeler_name;		
2548		int boot_type;		
2549		char *host_name;		
2550		int port_number;		
2551		{		
2552		u_long internet_address;		
2553		(void) strcpy((char *) boot_struct->lmsi, "LMSI");		
2554		#ifdef VMS		
2555		boot_struct->version = htons(EPRM_NVRAM_VERSION);		
2556		boot_struct->ethernet_packet_type = htons(LM_BOOT_PACKET_TYPE);		
2557		boot_struct->internet_socket_number = htons(port_number);		
2558		#else		
2559		boot_struct->version = EPRM_NVRAM_VERSION;		
2560		boot_struct->ethernet_packet_type = LM_BOOT_PACKET_TYPE;		
2561		boot_struct->internet_socket_number = port_number;		
2562		#endif		
2563		boot_struct->diagnostics = FALSE;		
2564		(void) memcpy((char *) boot_struct->boot_ethernet_address,		
2565		"0.0.0.0", 4);		
2566		boot_struct->boot_internet_address = 0;		
2567		if (MANUALBOOT != boot_type) {		
2568		{		
2569		if (FAILURE == get_internet_address(host_name, &internet_address)) {		
2570		(void) fprintf(stderr, "%s: unknown host: \"%s\"\n",		
2571		error_prefix, host_name);		
2572		return(FAILURE);		
2573		}		
2574		boot_struct->boot_internet_address = internet_address;		
2575		}		
2576		if (FAILURE == get_internet_address(modeler_name, &internet_address)) {		
2577		(void) fprintf(stderr, "%s: unknown modeler: \"%s\"\n",		
2578		error_prefix, modeler_name);		
2579		return(FAILURE);		
2580		}		
2581		boot_struct->modeler_internet_address = internet_address;		
2582		boot_struct->boot_machine_type = LM_BOOT_MACHINE;		
2583		boot_struct->autoboot = boot_type;		
2584		if (FAILURE == send_file(modeler_name, boot_struct))		
2585		return (FAILURE);		
2586		return(SUCCESS);		
2587		}		
2588		}		
2589		}		
2590		}		
2591		}		
2592		}		
2593		}		
2594		}		
2595		}		
2596		}		
2597		get_internet_address (node_name, internet_address)		
2598		{		
2599		char *node_name;		
2600		int *internet_address;		
2601		{		
2602		struct hostent *address_structure;		
2603		address_structure = (struct hostent *) gethostbyname (node_name);		
2604		if (address_structure == NULL) return(FAILURE);		
2605		*internet_address = *((unsigned int *) address_structure->h_addr;		
2606		return(SUCCESS);		
2607		}		
2608		}		
2609		}		
2610		}		
2611		send_file (modeler, boot_struct)		
2612		{		
2613		char *modeler;		
2614		BOOT_STRUCT *boot_struct;		
2615		{		
2616		long status;		
2617		LM_CHECK(LM_write(modeler, (u_long) LM_NVRAM_MEMORY, (u_long) 0,		
2618		(u_long) 0, (u_long) sizeof(BOOT_STRUCT),		
2619		(char *) boot_struct, &status));		
2620		}		
2621		return(SUCCESS);		
2622		}		
2623		}		
2624		}		
2625		}		
2626		}		
2627		u_exit ()		
2628		{		
2629		(void) signal(SIGCHUP, SIG_DFL);		
2630		(void) signal(SIGINT, SIG_DFL);		
2631		(void) signal(SIGTERM, SIG_DFL);		
2632		vprintf("%s\n", command_mode? "": "Ending utilities session...");		
2633		if (sigsetmask(0) == -1) {		
2634		perror("sigsetmask");		
2635		}		
2636		}		
2637		}		
2638		}		
2639		local_end_modeling_session();		
2640		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89 TIME 1:20:36 pm	PAGE # 23/57
LINE #	SOURCE TEXT			
2641	exit(0);			
2642	}			
2643	}			
2644	}			
2645	#define MAX_TRIES 3			
2646	}			
2647	check_password(modular_name)			
2648	char modular_name;			
2649	{			
2650	u_long count = 0;			
2651	long good_password;			
2652	long status;			
2653	char password[max_str];			
2654	char password_prompt[max_str];			
2655				
2656	status = lm_select_modelar(modular_name);			
2657	if (status != LM_SUCCESS)			
2658	return (SUCCESS);			
2659	/* If we can't select a modelar, keep going */			
2660				
2661	(void) lm_check (lm_inquire_modelar (LM_PASSWORD_CHECK, &good_password));			
2662				
2663	while (LM_FALSE == good_password) {			
2664	(void) sprintf(password_prompt, "Enter password for modelar \"%s\": ", modular_name);			
2665	prompt(password_prompt, password, sizeof(password));			
2666	(void) lm_check (lm_password(LM_ENTER, password));			
2667	(void) lm_check (lm_inquire_modelar (LM_PASSWORD_CHECK, &good_password));			
2668	if (MAX_TRIES <= ++count) {			
2669	fprintf(stderr, "%s: Too many password attempts\n", error_prefix);			
2670	return (FAILURE);			
2671	}			
2672				
2673	}			
2674	return (SUCCESS);			
2675	}			
2676	}			
2677				
2678				
2679	prompt (prompt_string, return_buffer, buffer_size)			
2680	char prompt_string;			
2681	char return_buffer;			
2682	int buffer_size;			
2683				
2684	{			
2685	#ifdef SUNOS_5			
2686	int (*save_stp_fn)(), handle_intr();			
2687	#else			
2688	void (*save_stp_fn)(), handle_intr();			
2689	#endif			
2690	extern jmp_buf jmpbuf;			
2691	jmp_buf jmpbuf;			
2692	jmp_buf save_jmpbuf = jmpbuf;			
2693				
2694	if (!setjmp(jmpbuf)) {			
2695				
2696	jmpbuf = (jmp_buf *) jmpbuf;			
2697	save_stp_fn = signal (SIGTSTP, lm_suspend);			
2698	raw ();			
2699	get_password(prompt_string, return_buffer, buffer_size);			
2700				
2701	} else {			
2702	jmpbuf = save_jmpbuf;			
2703	(void) signal (SIGTSTP, save_stp_fn);			
2704	cook();			
2705	(void) handle_intr(SIGINT);			
2706				
2707	jmpbuf = save_jmpbuf;			
2708				
2709	(void) signal (SIGTSTP, save_stp_fn);			
2710				
2711	cook();			
2712				
2713	}			
2714				
2715				
2716	get_password (prompt, reply, len)			
2717	char prompt;			
2718	char reply;			
2719	int len;			
2720				
2721	{			
2722	char c, *s;			
2723				
2724	s = reply;			
2725				
2726	(void) printf("%s", prompt);			
2727	(void) fflush(stdout);			
2728	#ifdef VMS			
2729	while (((c = getchar()) != LF) && (c != CR) && ((s-reply) < len)) {			
2730	#else			
2731	while (((c = vms_getchar()) != LF) && (c != CR) && ((s-reply) < len)) {			
2732	#endif			
2733	switch (c) {			
2734	case BS:			
2735	case DEL:			
2736	if (s > reply) --s;			
2737	break;			
2738	default:			
2739	if (!isprint(c)) {			
2740	*s++ = c;			
2741	} else {			
2742	putchar(ctrl(g));			
2743	}			
2744	}			
2745				
2746	putchar('\n');			
2747	*s = '\0';			
2748				
2749	}			
2750				
2751	#ifdef VMS			
2752	#include <io.h>			
2753	#include <stdio.h>			
2754	#include <string.h>			
2755				
2756	vms_getchar()			
2757	{			
2758	SDESCRIPTOR(device_name, "IT");			
2759				
2760	struct {			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/main.c	DATE 5/23/89	PAGE # 24/58
LINE #		SOURCE TEXT	TIME 1:20:36 pm	
2761		unsigned short condition_value,		
2762		unsigned short byte_count,		
2763		unsigned long status_bits,		
2764		} io_status,		
2765				
2766	static	unsigned short channel,		
2767	static	long event_flag = -1,		
2768	long	status,		
2769	char	buffer[100],		
2770				
2771				
2772		if (event_flag == -1) {		
2773		status = SYSASSIGN (sdevice_name, &channel, 0, 0);		
2774				
2775		if (status != SSS_NORMAL) {		
2776		LIBSIGNAL (status);		
2777		return(0);		
2778		}		
2779		status = LIBSGET_EF(&event_flag);		
2780				
2781		if (status != SSS_NORMAL) {		
2782		LIBSIGNAL (status);		
2783		return(0);		
2784		}		
2785				
2786		}		
2787				
2788		status = SYSQIOW (event_flag, channel, IOS_READVBLK + IOSM_NOECHO + IOSM_NOFILTR,		
2789		io_status, 0, 0, &buffer, 1, 0, 0, 0, 0);		
2790				
2791		if (status != SSS_NORMAL) {		
2792		LIBSIGNAL (status);		
2793		return(0);		
2794		}		
2795				
2796		return(&buffer);		
2797		}		
2798				
2799	#endif	/* VMS */		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/menus.c	DATE 5/23/89	PAGE # 1/59
LINE #	SOURCE TEXT			
1	/* SOURCE ID: menus.c rev. 3.1, 4/24/89 at 07:52:15 */			
2	#include <stdio.h>			
3	#include <ctype.h>			
4	#include <varargs.h>			
5	#include <setjmp.h>			
6	#include <signal.h>			
7	#include "common.h"			
8	#include "lm_afi.h"			
9	#include "menus.h"			
10	#include "lm_text.h"			
11				
12	static long current_depth = 0;			
13				
14	lm_display_menu (menu)			
15	LM_MENU *menu,			
16	{			
17	LM_MENU_ITEM *menu_item;			
18	long i, reply, quit_flag;			
19	char selection_string[max_str];			
20				
21	++current_depth;			
22				
23	while (LM_TRUE) {			
24	print_menu (menu);			
25	diag_get_selection (selection_string);			
26				
27	if (SUCCESS == built_in_commands (menu,			
28	selection_string, &quit_flag)) {			
29	if (quit_flag == LM_TRUE) {			
30	current_depth--;			
31	return(SUCCESS);			
32	}			
33	} else {			
34	reply = -1;			
35	menu_item = menu->menu_items;			
36	for (i = 0; i < menu->number_of_items; ++i) {			
37	if (0 == str_cmp (selection_string, (menu_item + i)->selection))			
38	reply = i;			
39	}			
40	if ((reply > -1) && !((menu_item + reply)->attributes & LM_no_select)) {			
41	menu->current_selection = reply;			
42	execute_routine (menu);			
43	if (menu->menu_items[menu->current_selection].attributes & LM_automatic_quit) {			
44	current_depth--;			
45	return(SUCCESS);			
46	}			
47	} else (void) printf("Invalid selection: %s\n", selection_string);			
48	}			
49	}			
50				
51	built_in_commands (menu, selection_string, quit_flag)			
52	LM_MENU *menu;			
53	char selection_string;			
54	long quit_flag;			
55	{			
56	LM_MENU_ITEM *menu_item;			
57	long i;			
58				
59	quit_flag = LM_FALSE;			
60				
61	if (0 == str_cmp(selection_string, "quit",			
62	strlen(selection_string))) {			
63	quit_flag = LM_TRUE;			
64	return(SUCCESS);			
65	}			
66	else if (0 == str_cmp(selection_string, "help", strlen(selection_string))) {			
67	menu_item = menu->menu_items;			
68	if ((menu_item->help_string == NULL)			
69	{ (void) printf("Sorry, no help available\n");			
70	else {			
71	lm_get_input ("Menu item: ", selection_string, max_str);			
72	for (i = 0; i < menu->number_of_items; ++i) {			
73	if (0 == str_cmp(selection_string, (menu_item + i)->selection)) {			
74	if ((menu_item->help_string == NULL) {			
75	(void) printf("Sorry, no help available\n");			
76	} else {			
77	(void) printf ("%s\n", (menu_item + i)->help_string);			
78	lm_get_input ("Press return to continue: ", selection_string, max_str);			
79	}			
80	}			
81	return(SUCCESS);			
82	}			
83	else if (0 == str_cmp(selection_string, "exit", 1)) {			
84	u_exit(); /* never returns */			
85	return(SUCCESS);			
86	}			
87	else if (0 == str_cmp(selection_string, "field_engineering")) {			
88	lm_promote();			
89	return(SUCCESS);			
90	}			
91	return(FAILURE);			
92	}			
93				
94	int Fe_mode = FALSE;			
95				
96	lm_promote()			
97	{			
98	Fe_mode = TRUE;			
99	}			
100				
101	print_menu (menu)			
102	LM_MENU *menu,			
103	{			
104	LM_MENU_ITEM *menu_list;			
105	long i;			
106				
107	(void) fprintf(stderr, "\n");			
108				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/menus.c	DATE 5/23/89	PAGE # 2/60
SOURCE TEXT				
LINE #	SOURCE TEXT			
121	(void) fprintf(stderr, " %s\n", menu->title);			
122	(void) fprintf(stderr, "\n");			
123	menu_list = menu->menu_items;			
124	for (i=0; i<menu->number_of_items; i++, menu_list++) {			
125	if (!((menu_list->attributes & LM_no_display))			
126	(void) fprintf(stderr, "%4s %s\n", menu_list->selection, menu_list->menu_text);			
127	}			
128	menu_list = menu->menu_items;			
129	if ((menu_list->help_string != NULL)			
130	(void) fprintf(stderr, "%4s %s\n", "h", " Help");			
131	(void) fprintf(stderr, "%4s %s\n", "q", " Quit");			
132	}			
133	diag_get_selection (string_reply)			
134	char *string_reply;			
135	{			
136	(void) fprintf(stderr, "\n");			
137	do {			
138	lm_get_input("selection: ", string_reply, max_str);			
139	clear_input (string_reply);			
140	} while (strlen(string_reply) == 0);			
141	(void) fprintf(stderr, "\n");			
142	}			
143	/* interrupts while executing should return user to the menu */			
144	jmp_buf *Jmpbuf;			
145	handle_intr(sig)			
146	int sig;			
147	{			
148	if (sig == 0) sig = 1;			
149	#ifdef lint			
150	longjmp(Jmpbuf, sig); /* sig better not be 0! why not? because			
151	some's will pass the 0 back as the			
152	return value to setjmp when you do			
153	the longjmp. That is a bad thing */			
154	#else			
155	/* longjmp */(Jmpbuf, sig); /* lint is stupid I'm smarter */			
156	#endif			
157	}			
158	enroute routine (menu)			
159	LM_MENU *menu;			
160	{			
161	#ifdef SIGNAL_5			
162	int (*save_intr_fcn)();			
163	#else			
164	void (*save_intr_fcn)();			
165	#endif			
166	jmp_buf Jmpbuf; *save_Jmpbuf = Jmpbuf;			
167	if (!((menu->menu_items[menu->current_selection].attributes			
168	& LM_no_enroute)) {			
169	if (!setjmp(Jmpbuf)) {			
170	Jmpbuf = (jmp_buf *)Jmpbuf;			
171	save_intr_fcn = signal(SIGINT, handle_intr);			
172	if (menu->menu_items[menu->current_selection].attributes			
173	& LM_routine)			
174	local_begin_modeling_session();			
175	(*menu->menu_items[menu->current_selection].action_routine)			
176	(menu->menu_items[menu->current_selection].user_data);			
177	}			
178	local_end_modeling_session();			
179	Jmpbuf = save_Jmpbuf;			
180	(void) signal(SIGINT, save_intr_fcn);			
181	}			
182	}			
183	str_cmp (s1, s2)			
184	char *s1, *s2;			
185	{			
186	long i, j;			
187	char c1, c2;			
188	{			
189	i = strlen(s1);			
190	if (strlen(s2)>i) i = strlen(s2);			
191	for (i=0; i<i, i++) {			
192	c1 = s1[i];			
193	if (isupper(c1)) c1 = tolower(c1);			
194	c2 = s2[i];			
195	if (isupper(c2)) c2 = tolower(c2);			
196	if (c1 != c2) return (c1 - c2);			
197	}			
198	return(0);			
199	}			
200	strncmp (s1, s2, max)			
201	char *s1, *s2;			
202	int max;			
203	{			
204	long i, j;			
205	char c1, c2;			
206	{			
207	i = strlen(s1);			
208	if (strlen(s2)>i) i = strlen(s2);			
209	if (max<i) i = max;			
210	for (i=0; i<i, i++) {			
211	c1 = s1[i];			
212	if (isupper(c1)) c1 = tolower(c1);			
213	c2 = s2[i];			
214	if (isupper(c2)) c2 = tolower(c2);			
215	if (c1 != c2) return (c1 - c2);			
216	}			
217	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/menus.c	DATE 5/23/89	PAGE # 3/61
LINE #		SOURCE TEXT		
241		return(0);		
242				
243				
244				
245		static int Modeling_session_in_progress = 0;		
246				
247		local_begin_modeling_session()		
248		{		
249		extern char *User_name;		
250		extern char Host_name[];		
251				
252		if (Modeling_session_in_progress == 0) {		
253		if (lm_check(lm_begin_modeling_session(LM_LOGIC_SIMULATOR, Host_name, User_name))		
254		!= LM_ERROR) {		
255		Modeling_session_in_progress = 1;		
256		}		
257		}		
258				
259				
260				
261		local_end_modeling_session()		
262		{		
263		if (Modeling_session_in_progress) {		
264		Modeling_session_in_progress = 0;		
265		lm_end_modeling_session();		
266		}		
267		}		

Copyright 1989 Logic Modeling Systems		HEADER FILE lm/menus.h	DATE 5/23/89 TIME 1:20:39 pm	PAGE # 1/62
LINE #		HEADER TEXT		
1		/* SCCS ID: menus.h rev 3.1. 4/24/89 at 09:09:20 */		
2		/* Attributes fields */		
3		#define LM_another_menu 0x0001 /* menu item is really a sub menu */		
4		#define LM_routine 0x0002 /* menu item is a diagnostic routine */		
5		#define LM_no_select 0x0004 /* menu item will not be selectable */		
6		#define LM_no_display 0x0008 /* menu item will not be displayed */		
7		#define LM_no_execute 0x0010 /* menu item will not be executed */		
8		/* when "all" is selected */		
9		#define LM_production_only 0x0020 /* menu item will display/execute		
10		in "production" only */		
11		#define LM_no_repeat 0x0040 /* menu item cannot be repeated */		
12		#define LM_automatic_quit 0x0080 /* quits after one selection */		
13		#define LM_no_summary 0x1000 /* menu item will not display summary */		
14		#define LM_null (char *) 0		
15		/* Status word bit setting macros */		
16		#define LM_MENU_ITEM		
17		typedef struct {		
18		char *selection; /* the character string for selecting the item */		
19		char *menu_text; /* a description of the menu item */		
20		int (* action_routine)(); /* the address of the routine to run */		
21		int attributes; /* and attribute bits above */		
22		char *user_data; /* pointer to user data passed to user routine */		
23		char *help_string;		
24		} LM_MENU_ITEM;		
25		#define LM_MENU		
26		typedef struct {		
27		char *title; /* the title of the menu */		
28		short number_of_items; /* number of items in the menu */		
29		short current_selection; /* index into menu_items */		
30		LM_MENU_ITEM *menu_items; /* address of the menu structure array */		
31		} LM_MENU;		
32		#define MAX_STR 255		
33		#define MAX_DEPTH 20		
34		#define RS 8		
35		#define CR 13		
36		#define LF 10		
37		#define DEL 127		
38		#define CTRL(c) ('c' & 0x1f)		
39		extern int lm_display_menu();		
40		char lm_get_key();		
41		char *lm_get_line();		
42		extern int Fe_mode;		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/tmgfiltr.c	DATE 5/23/89	PAGE # 1/63
LINE #		SOURCE TEXT		
1		/* SCCS ID: tmgfiltr.c rev 1.2 5/3/89 at 08:36:15 */		
2		#include <stdio.h>		
3		#include <ctype.h>		
4		#include "common.h"		
5		#define INITIAL_DELAY_TABLE 16384		
6		#define INCREMENTAL_DELAY_TABLE 8192		
7				
8		#define LOW 0		
9		#define HIGH 1		
10		#define FLOAT 2		
11				
12		#define STATE(state) s_array[state]		
13				
14		#define INVALID_COMPOSITE_INDEX 0xffffffff		
15				
16		#define NUMERIC_SUFFIX 0x01		
17				
18		typedef struct Delay {		
19		u_long flag;		
20		char *text;		
21		u_long numeric;		
22		u_long min_range;		
23		u_long max_range;		
24		float total;		
25		float squared_total;		
26		u_long number_of_elements;		
27		u_long composite_index;		
28		char *f_name;		
29		u_long f_state;		
30		char *t_name;		
31		u_long t_state;		
32		u_long minimum;		
33		u_long maximum;		
34		} DELAY;		
35				
36		static int heuristic;		
37				
38		static FILE *fpin;		
39		static FILE *fpout;		
40				
41		static u_long delay_table_size = 0;		
42		static u_long number_of_delays = 0;		
43		static DELAY *delay_table = (DELAY *)NULL;		
44				
45		static char *s_array[] = { "low", "high", "float" };		
46				
47		#define vfprintf (void) fprintf		
48				
49		#define MAXNAME 256		
50		#define MAXLINE 1024		
51				
52		#define MAX_INTEGER 0xffffffff		
53		#define MIN_INTEGER 0x0		
54				
55		#ifdef DEBUG2		
56		main(argc, argv)		
57		int		
58		char **argv,		
59		{		
60		FILE *fp;		
61		int		
62		i;		
63		int bus_mode;		
64		int error_condition;		
65				
66		if (argc < 2) {		
67		vfprintf(stderr, "usage: ts file [file ...]\n", argv[0]);		
68		exit(EXIT_FATAL_ERROR);		
69		}		
70		bus_mode = FALSE;		
71		error_condition = FALSE;		
72		for(i=1, i(argc, ++i) {		
73		if ((fp=fopen(argv[i], "r")) == NULL) {		
74		error_condition = TRUE;		
75		vfprintf(stderr, "error: failure to open file ts for reading\n", argv[i]);		
76		}		
77		else {		
78		if (lm_filter_timing(fp) != SUCCESS) {		
79		error_condition = TRUE;		
80		}		
81		(void) fclose(fp);		
82		}		
83		if (error_condition == FALSE) {		
84		if (lm_done_with_filter_timing(bus_mode, stdout) != SUCCESS) {		
85		error_condition = TRUE;		
86		}		
87		}		
88		if (error_condition == TRUE)		
89		exit(EXIT_FATAL_ERROR);		
90		else exit(EXIT_NO_ERROR);		
91		}		
92		#endif /* DEBUG2 */		
93				
94		lm_filter_timing(file_pointer_in)		
95		FILE *file_pointer_in;		
96		{		
97		extern char *calloc();		
98				
99		fpin = file_pointer_in;		
100				
101		if ((delay_table == (DELAY *)NULL) (delay_table_size == 0)) {		
102		delay_table_size = INITIAL_DELAY_TABLE;		
103		delay_table = (DELAY *) calloc((unsigned)delay_table_size, sizeof(DELAY));		
104		if (delay_table == (DELAY *) NULL) {		
105		vfprintf(stderr, "error: failure to allocate host memory to store delay table\n");		
106		return(FAILURE);		
107		}		
108		}		
109		if (read_in_information() != SUCCESS) {		
110				

<div> <div>Copyright 1989</div> <div>Logic Modeling Systems</div> </div>	<div> <div>SOURCE PROGRAM</div> <div>lm/tmgfiltr.c</div> </div>		<div> <div>DATE5/23/89</div> <div>TIME1:20:39 pm</div> </div>	<div> <div>PAGE #</div> <div>2/64</div> </div>		
	SOURCE TEXT					
	LINE #					
121	return(FAILURE);					
122	}					
123	return(SUCCESS);					
124	}					
125	}					
126	}					
127	}					
128	lm_done_with_filter_timing(bus_mode, file_pointer_out)					
129	int					
130	FILE					
131	{					
132	u_long i;					
133	int					
134	error_condition;					
135	{					
136	ipout = file_pointer_out;					
137	heuristic = bus_mode;					
138	error_condition = FALSE;					
139	if (process_information() != SUCCESS) {					
140	error_condition = TRUE;					
141	}					
142	else {					
143	if (print_results() != SUCCESS) {					
144	error_condition = TRUE;					
145	}					
146	}					
147	}					
148	}					
149	}					
150	for(i=0; i<(number_of_delays; ++i) {					
151	if (delay_table[i].text != (char *)NULL)					
152	(void) free(delay_table[i].text);					
153	if (delay_table[i].f_name != (char *)NULL)					
154	(void) free(delay_table[i].f_name);					
155	if (delay_table[i].t_name != (char *)NULL)					
156	(void) free(delay_table[i].t_name);					
157	}					
158	if (delay_table != (DELAY *)NULL)					
159	(void) free((char *)delay_table);					
160	number_of_delays = 0;					
161	delay_table_size = 0;					
162	delay_table = (DELAY *)NULL;					
163	if (error_condition == TRUE)					
164	return(FAILURE);					
165	else return(SUCCESS);					
166	}					
167	}					
168	}					
169	}					
170	}					
171	static					
172	read_in_information()					
173	{					
174	int					
175	line_no;					
176	int					
177	error_condition;					
178	int					
179	max_errors_to_print;					
180	{					
181	int					
182	timestamp;					
183	int					
184	current_timestamp;					
185	char					
186	delay[MAXNAME];					
187	from[MAXNAME];					
188	f_state[MAXNAME];					
189	t_state[MAXNAME];					
190	t_name[MAXNAME];					
191	equals[MAXNAME];					
192	mis[MAXNAME];					
193	max[MAXNAME];					
194	{					
195	char					
196	*ptr;					
197	char					
198	line[MAXLINE];					
199	{					
200	line_no = 0;					
201	error_condition = 0;					
202	current_timestamp = 0;					
203	max_errors_to_print = 10;					
204	while(!fgets(line, MAXLINE, fpin) != NULL) {					
205	++line_no;					
206	{					
207	ptr = line + strlen(line, "\t");					
208	if (strcmp(ptr, "device", 6) == 0) {					
209	continue;					
210	}					
211	else if (*ptr == '\n') {					
212	continue;					
213	}					
214	rtm-second(line, "%d %s %s (%s					

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/tmgfiltr.c	DATE 5/23/89	PAGE # 3/65
LINE #		SOURCE TEXT		
239		if (strcmp(to, "to") != 0) {		
240		vfprintf(stderr, "error: expected a token \"to\", instead got a token \"%s\" on line number %d:\n\tts", to, line_no, 1		
241		line);		
242		++error_condition;		
243		continue;		
244		}		
245		if (strcmp(equals, "=") != 0) {		
246		vfprintf(stderr, "error: expected a token \"=\", instead got a token \"%s\" on line number %d:\n\tts", equals, line_no		
247		, line);		
248		++error_condition;		
249		continue;		
250		}		
251		if (timestamp > current_timestamp) {		
252		current_timestamp = timestamp;		
253		}		
254		else if (timestamp < current_timestamp) {		
255		vfprintf(stderr, "error: decreasing time on line number %d:\n\tts", line_no, line);		
256		++error_condition;		
257		continue;		
258		}		
259		delay_table[number_of_delays].composite_index = INVALID_COMPOSITE_INDEX;		
260		}		
261		if (nameenter(delay_table[number_of_delays].f_name, f_name) != SUCCESS) {		
262		++error_condition;		
263		continue;		
264		}		
265		if (nameenter(delay_table[number_of_delays].t_name, t_name) != SUCCESS) {		
266		++error_condition;		
267		continue;		
268		}		
269		intenter(delay_table[number_of_delays].minimum, min);		
270		intenter(delay_table[number_of_delays].maximum, max);		
271		}		
272		if (state_enter(delay_table[number_of_delays].f_state, f_state) != SUCCESS) {		
273		vfprintf(stderr, "error: expected a token \"low\", \"high\", or \"float\", instead got a token \"%s\" on line number %d:\n\tts", f_state, line_no, line);		
274		++error_condition;		
275		continue;		
276		}		
277		if (state_enter(delay_table[number_of_delays].t_state, t_state) != SUCCESS) {		
278		vfprintf(stderr, "error: expected a token \"low\", \"high\", or \"float\", instead got a token \"%s\" on line number %d:\n\tts", t_state, line_no, line);		
279		++error_condition;		
280		continue;		
281		}		
282		}		
283		/* Initialize all other flags */		
284		delay_table[number_of_delays].flag = 0;		
285		delay_table[number_of_delays].text = (char *)NULL;		
286		delay_table[number_of_delays].numeric = 0;		
287		delay_table[number_of_delays].min_range = 0;		
288		delay_table[number_of_delays].max_range = 0;		
289		delay_table[number_of_delays].total = 0.0;		
290		delay_table[number_of_delays].squared_total = 0.0;		
291		delay_table[number_of_delays].number_of_elements = 0;		
292		}		
293		if ((number_of_delays >= delay_table_size) {		
294		if (increase_delay_table_size() != SUCCESS) {		
295		return(FAILURE);		
296		}		
297		}		
298		}		
299		if (error_condition > max_errors_to_print) {		
300		vfprintf(stderr, "error: too many errors found: %d, giving up", error_condition);		
301		return(FAILURE);		
302		}		
303		}		
304		}		
305		if (error_condition > 0)		
306		return(FAILURE);		
307		else return(SUCCESS);		
308		}		
309		}		
310		}		
311		}		
312		static		
313		nameenter(dest, src)		
314		char **dest;		
315		char *src;		
316		{		
317		int last;		
318		char *ptr;		
319		char temp(MAXNAME);		
320		{		
321		last = strlen(src) - 1;		
322		}		
323		/* If "src" is a legal name or already has quotes, use "src" */		
324		if (((isalpha(src)) ("src" == "")) && (rest_of_string_is_alnum(src) == TRUE)		
325		("src" == "" && src[last] == '\\') ("src" == "" && src[last] == '"')) {		
326		{		
327		if (streenter(dest, src) != SUCCESS) {		
328		return(FAILURE);		
329		}		
330		}		
331		else {		
332		/* else, quote the name */		
333		temp[0] = '\\';		
334		for(ptr=temp; *src!='\\' && *src!='\"'; ++ptr) {		
335		if (*src == '\\') /* double up single quotes */		
336		++ptr = '\\';		
337		++ptr = *src++;		
338		}		
339		++ptr = '\\';		
340		++ptr = '\\0';		
341		if (streenter(dest, temp) != SUCCESS) {		
342		return(FAILURE);		
343		}		
344		}		
345		}		
346		return(SUCCESS);		
347		}		
348		}		
349		}		
350		}		
351		static		
352		rest_of_string_is_alnum(src)		
353		char *src;		
354		{		

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM	DATE	PAGE #
	lm/tmgfiltr.c	5/23/89	4/66
		TIME	1:20:39 pm

LINE #	SOURCE TEXT
352	register char c;
353	while((c = *++src) != '\0') {
354	if (c != ' ' && !isalnum(c))
355	return(FALSE);
356	}
357	return(TRUE);
358	}
359	static
360	strnter(dest, src)
361	char *dest,
362	char *src;
363	{
364	if ((*dest = malloc((strlen(src)+1))) == (char *) NULL) {
365	vfprintf(stderr, "error: failure to allocate host memory to store delay information\n");
366	return(FALSE);
367	}
368	(void) strcpy(dest, src);
369	return(SUCCESS);
370	}
371	static
372	intnter(dest, src)
373	u_long *dest,
374	char *src;
375	{
376	extern double atof();
377	
378	*dest = (u_long) (atof(src) * 1000.0);
379	}
380	static
381	state_enter(dest, src)
382	u_long *dest,
383	char *src;
384	{
385	int i;
386	
387	for(i=0; i<(sizeof(s_array)/sizeof(char *)); ++i) {
388	if (strcmp(s_array[i], src) == 0) {
389	*dest = i;
390	return(SUCCESS);
391	}
392	return(FALSE);
393	}
394	static
395	increase_delay_table_size()
396	{
397	DELAY *ptr;
398	extern char *realloc();
399	
400	delay_table_size += INCREMENTAL_DELAY_TABLE;
401	ptr = (DELAY *) realloc((char *)delay_table, (unsigned)(delay_table_size*sizeof(DELAY)));
402	if (ptr == (DELAY *) NULL) {
403	vfprintf(stderr, "error: failure to allocate host memory for internal delay table\n");
404	return(FALSE);
405	}
406	delay_table = ptr;
407	return(SUCCESS);
408	}
409	static
410	process_information()
411	{
412	int compare();
413	
414	#ifdef VMS
415	vms_qsort((char *)delay_table, (int) number_of_delays, sizeof(DELAY), compare);
416	#else
417	qsort((char *)delay_table, (int) number_of_delays, sizeof(DELAY), compare);
418	#endif
419	/* VMS */
420	if (collapse_same_delays() != SUCCESS) {
421	return(FALSE);
422	}
423	if (create_composite_delays() != SUCCESS) {
424	return(FALSE);
425	}
426	if (heuristic == TRUE) {
427	if (create_business() != SUCCESS) {
428	return(FALSE);
429	}
430	return(SUCCESS);
431	}
432	static
433	compare(delay1, delay2)
434	register DELAY *delay1,
435	register DELAY *delay2;
436	{
437	int rtn;
438	
439	if (heuristic == TRUE) {
440	/* Note that the order is extremely important to the bus creation routine. */
441	rtn = strcmp(delay1->f_name, delay2->f_name);
442	if (rtn != 0) return(rtn);

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/tmgfilt.r.c	DATE 5/23/89	PAGE # 5/67
LINE #		SOURCE TEXT		
475		if (delay1->f_state < delay2->f_state)		
476		return(-1);		
477		else if (delay1->f_state > delay2->f_state)		
478		return(1);		
479				
480		if (delay1->t_state < delay2->t_state)		
481		return(-1);		
482		else if (delay1->t_state > delay2->t_state)		
483		return(1);		
484				
485		rta = strcmp(delay1->t_name, delay2->t_name);		
486		if (rta != 0) return(rta);		
487				
488		}		
489		else {		
490		rta = strcmp(delay1->f_name, delay2->f_name);		
491		if (rta != 0) return(rta);		
492				
493		if (delay1->f_state < delay2->f_state)		
494		return(-1);		
495		else if (delay1->f_state > delay2->f_state)		
496		return(1);		
497				
498		rta = strcmp(delay1->t_name, delay2->t_name);		
499		if (rta != 0) return(rta);		
500				
501		if (delay1->t_state < delay2->t_state)		
502		return(-1);		
503		else if (delay1->t_state > delay2->t_state)		
504		return(1);		
505				
506		}		
507		return(0);		
508				
509				
510				
511		static		
512		collapse_same_delays()		
513		{		
514		u_long i;		
515		u_long base_i;		
516		u_long current_composite_index;		
517				
518		base_i = 0;		
519		current_composite_index = 1;		
520				
521		/* Setup the first element. */		
522		delay_table[base_i].total =		
523		(float)delay_table[base_i].minimum + (float)delay_table[base_i].maximum;		
524		delay_table[base_i].squared_total =		
525		(float)delay_table[base_i].minimum * (float)delay_table[base_i].minimum +		
526		(float)delay_table[base_i].maximum * (float)delay_table[base_i].maximum;		
527		delay_table[base_i].number_of_elements = 2;		
528		delay_table[base_i].composite_index = current_composite_index;		
529				
530		for(i=1, i<number_of_delays, ++i) {		
531		/* Note that composite_index was set to INVALID above. */		
532		if (same_to_and_from_name_and_state(base_i, i) == TRUE) {		
533		delay_table[base_i].composite_index = current_composite_index;		
534				
535		delay_table[base_i].total +=		
536		(float)delay_table[i].minimum + (float)delay_table[i].maximum;		
537		delay_table[base_i].squared_total +=		
538		(float)delay_table[i].minimum * (float)delay_table[i].minimum +		
539		(float)delay_table[i].maximum * (float)delay_table[i].maximum;		
540		delay_table[base_i].number_of_elements += 2;		
541				
542		if (delay_table[base_i].minimum > delay_table[i].minimum)		
543		delay_table[base_i].minimum = delay_table[i].minimum;		
544		if (delay_table[base_i].maximum < delay_table[i].maximum)		
545		delay_table[base_i].maximum = delay_table[i].maximum;		
546				
547		}		
548		else {		
549		base_i = i;		
550		++current_composite_index;		
551				
552		delay_table[i].total =		
553		(float)delay_table[i].minimum + (float)delay_table[i].maximum;		
554		delay_table[i].squared_total =		
555		(float)delay_table[i].minimum * (float)delay_table[i].minimum +		
556		(float)delay_table[i].maximum * (float)delay_table[i].maximum;		
557		delay_table[i].number_of_elements = 2;		
558		delay_table[i].composite_index = current_composite_index;		
559				
560		}		
561				
562		return(SUCCESS);		
563				
564				
565				
566				
567		static		
568		same_to_and_from_name_and_state(i, next_i)		
569		{		
570		u_long i;		
571		u_long next_i;		
572		{		
573		if (next_i >= number_of_delays) {		
574		return(FALSE);		
575		}		
576		if ((delay_table[i].f_state == delay_table[next_i].f_state) &&		
577		(delay_table[i].t_state == delay_table[next_i].t_state) &&		
578		(strcmp(delay_table[i].f_name, delay_table[next_i].f_name) == 0) &&		
579		(strcmp(delay_table[i].t_name, delay_table[next_i].t_name) == 0))		
580		{		
581		return(TRUE);		
582		}		
583		return(FALSE);		
584				
585				
586				
587				
588		static		
589		create_composite_delays()		
590		{		
591		u_long i;		
592		u_long base_i;		
593		u_long current_composite_index;		
594				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/tmgfiltr.c	DATE 5/23/89	PAGE # 6/68
LINE #		SOURCE TEXT		
192		base_i = 0;		
193		current_composite_index = 1;		
194		for(i=0, i<number_of_delays, ++i) {		
195		if (delay_table[i].composite_index == INVALID_COMPOSITE_INDEX) {		
196		continue;		
197		}		
198		delay_table[base_i].composite_index = current_composite_index;		
199		if (same_from_name_and_state(base_i, i) == TRUE) {		
200		delay_table[i].composite_index = current_composite_index;		
201		}		
202		else {		
203		base_i = i;		
204		++current_composite_index;		
205		}		
206		}		
207		return(SUCCESS);		
208		}		
209		static		
210		same_from_name_and_state(i, next_i)		
211		u_long i,		
212		u_long next_i,		
213		{		
214		if (next_i >= number_of_delays) {		
215		return(FALSE);		
216		}		
217		if ((delay_table[i].f_state == delay_table[next_i].f_state) &&		
218		(strcmp(delay_table[i].f_name, delay_table[next_i].f_name) == 0))		
219		{		
220		return(TRUE);		
221		}		
222		return(FALSE);		
223		}		
224		static		
225		create_bases()		
226		{		
227		if (find_numeric_suffices() != SUCCESS) {		
228		return(FALSE);		
229		}		
230		if (collapse_bases() != SUCCESS) {		
231		return(FALSE);		
232		}		
233		return(SUCCESS);		
234		}		
235		static		
236		find_numeric_suffices()		
237		{		
238		u_long i,		
239		char *ptr,		
240		char temp[BUFSIZ];		
241		{		
242		for(i=0, i<number_of_delays, ++i) {		
243		(void)strcpy(temp, delay_table[i].t_name);		
244		delay_table[i].flag = 0;		
245		delay_table[i].text = (char *)NULL;		
246		delay_table[i].numeric = 0;		
247		if (isdigit(temp[strlen(temp)-1])) {		
248		delay_table[i].flag = NUMERIC_SUFFIX;		
249		for(ptr = &temp[strlen(temp)-1]; isdigit(*ptr); --ptr)		
250		/* Just back up */		
251		delay_table[i].numeric = atoi(ptr+1);		
252		delay_table[i].min_range = delay_table[i].numeric;		
253		delay_table[i].max_range = delay_table[i].numeric;		
254		++ptr = '\0'; /* Fresh numeric suffix */		
255		if (strcmp(&delay_table[i].text, temp) != SUCCESS) {		
256		return(FALSE);		
257		}		
258		}		
259		return(SUCCESS);		
260		}		
261		static		
262		collapse_bases()		
263		{		
264		u_long i,		
265		u_long base_i,		
266		{		
267		base_i = 0; /* Start with element zero */		
268		for(i=0, i<number_of_delays, ++i) {		
269		if (delay_table[i].composite_index == INVALID_COMPOSITE_INDEX) {		
270		continue;		
271		}		
272		if ((delay_table[base_i].flag == NUMERIC_SUFFIX) &&		
273		(delay_table[i].flag == NUMERIC_SUFFIX) &&		
274		(delay_table[base_i].max_range+1 == delay_table[i].numeric) &&		
275		(delay_table[base_i].f_state == delay_table[i].f_state) &&		
276		(delay_table[base_i].t_state == delay_table[i].t_state) &&		
277		(strcmp(delay_table[base_i].text, delay_table[i].text) == 0))		
278		{		
279		delay_table[base_i].max_range = delay_table[i].numeric;		
280		delay_table[base_i].total += delay_table[i].total;		
281		delay_table[base_i].squared_total += delay_table[i].squared_total;		
282		delay_table[base_i].number_of_elements += delay_table[i].number_of_elements;		
283		if (delay_table[base_i].minimum > delay_table[i].minimum)		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM lm/tmgfilt.c	DATE 5/23/89 TIME 1:20:39 pm	PAGE # 7/69
LINE #	SOURCE TEXT			
715	delay_table[base_i].minimum = delay_table[i].minimum;			
716	if (delay_table[base_i].maximum < delay_table[i].maximum)			
717	delay_table[base_i].maximum = delay_table[i].maximum;			
718	delay_table[i].composite_index = INVALID_COMPOSITE_INDEX;			
719	}			
720	else {			
721	base_i = i;			
722	}			
723	}			
724	return(SUCCESS);			
725	}			
726	}			
727	}			
728	}			
729	#define PREFERRED_EQUAL_WIDTH 24			
730	#define PREFERRED_COMMENT_WIDTH 40			
731	static			
732	print_results()			
733	{			
734	int len;			
735	int spaces;			
736	u_long i;			
737	u_long current_composite_index;			
738	extern double standard_deviation();			
739	current_composite_index = INVALID_COMPOSITE_INDEX;			
740	for(i=0; i<number_of_delays; ++i) {			
741	if (delay_table[i].composite_index == INVALID_COMPOSITE_INDEX) {			
742	continue;			
743	}			
744	if (current_composite_index != delay_table[i].composite_index) {			
745	current_composite_index = delay_table[i].composite_index;			
746	vfprintf(stdout, "delay from %s(%s)\n",			
747	STATE(delay_table[i].t_state),			
748	delay_table[i].t_name			
749);			
750	}			
751	if (delay_table[i].min_range == delay_table[i].max_range) {			
752	len = fprintf(stdout, " to %s(%s)",			
753	STATE(delay_table[i].t_state),			
754	delay_table[i].t_name			
755);			
756	}			
757	else {			
758	len = fprintf(stdout, " to %s(%s[%d:%d])",			
759	STATE(delay_table[i].t_state),			
760	delay_table[i].t_name,			
761	delay_table[i].min_range,			
762	delay_table[i].max_range			
763);			
764	}			
765	if (len < PREFERRED_EQUAL_WIDTH)			
766	spaces = PREFERRED_EQUAL_WIDTH - len;			
767	else spaces = 8 - len%8;			
768	len += fprintf(stdout, "%s= %g, %g",			
769	spaces, " ",			
770	(((float)delay_table[i].minimum) / 1000.0),			
771	(((float)delay_table[i].maximum) / 1000.0)			
772);			
773	if (len < PREFERRED_COMMENT_WIDTH)			
774	spaces = PREFERRED_COMMENT_WIDTH - len;			
775	else spaces = 8 - len%8;			
776	vfprintf(stdout, "%s(Samples=%ld, Mean=%4.1f, Std dev=%6.2f)\n",			
777	spaces, " ",			
778	delay_table[i].number_of_elements,			
779	delay_table[i].total / delay_table[i].number_of_elements / 1000.0,			
780	(float) (standard_deviation(i) / 1000.0)			
781);			
782	}			
783	return(SUCCESS);			
784	}			
785	}			
786	static double			
787	standard_deviation(index)			
788	{			
789	double total;			
790	double squared;			
791	double elements;			
792	extern double fabs();			
793	extern double sqrt();			
794	{			
795	/*			
796	Standard deviation =			
797	square_root((summation of (xi^2) - (sum(xi)^2)/(N-1))			
798	or			
799	square_root((summation of (xi^2) - ((total^2)/N)/(N-1))			
800	or			
801	square_root(squared_total - ((total*total)/N)/(N-1))			
802	*/			
803	total = (double) delay_table[index].total;			
804	squared = (double) delay_table[index].squared_total;			
805	elements = (double) delay_table[index].number_of_elements;			
806	return(sqrt(fabs(squared - total * total / elements) / (elements - 1.0)));			
807	}			
808	static			
809	strcmp(str1, str2)			
810	register char *str1;			
811	register char *str2;			
812	{			
813	/* This strcmp function orders subscripted names correctly. I.e.			
814	"A31" is greater than "A3"			

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM lm/tmgfiltr.c	DATE	5/23/89	PAGE #	8/70
		TIME	1:20:39 pm		

LINE #	SOURCE TEXT
812	/* stolen from Robert's sfi/utill.c
813	*/
814	
815	u_long str1_len;
816	u_long str2_len;
817	u_long subscript1;
818	u_long subscript2;
819	char *str1_subscript;
820	char *str2_subscript;
821	char *limit;
822	char save_char1;
823	char save_char2;
824	u_char str1_has_subscript = FALSE;
825	u_char str2_has_subscript = FALSE;
826	
827	str1_len = strlen(str1);
828	str2_len = strlen(str2);
829	
830	if (!isdigit(str1[str1_len - 1]))
831	str1_has_subscript = TRUE;
832	
833	if (!isdigit(str2[str2_len - 1]))
834	str2_has_subscript = TRUE;
835	
836	if (! ((str1_has_subscript == TRUE) && (str2_has_subscript == TRUE))) {
837	return(strcmp(str1, str2));
838	}
839	
840	/* Scan from the end of the string. At the end of the while loop,
841	* str1_subscript will point to the last non-digit character, or
842	* point to the '\0' at the end of the string if the string is all digits.
843	*/
844	str1_subscript = &str1[str1_len - 1];
845	limit = &str1[0];
846	while ((long)str1_subscript >= (long)limit) {
847	if (!isdigit(*str1_subscript))
848	--str1_subscript;
849	else
850	break;
851	}
852	++str1_subscript;
853	
854	str2_subscript = &str2[str2_len - 1];
855	limit = &str2[0];
856	while ((long)str2_subscript >= (long)limit) {
857	if (!isdigit(*str2_subscript))
858	--str2_subscript;
859	else
860	break;
861	}
862	++str2_subscript;
863	
864	save_char1 = *str1_subscript;
865	save_char2 = *str2_subscript;
866	
867	*str1_subscript = '\0';
868	*str2_subscript = '\0';
869	
870	if (strcmp(str1, str2) == 0) {
871	/* The non-digit part of the string matches --> look at the subscript */
872	
873	*str1_subscript = save_char1;
874	*str2_subscript = save_char2;
875	
876	(void)memset(str1_subscript, "\0", &subscript1);
877	(void)memset(str2_subscript, "\0", &subscript2);
878	
879	if (subscript1 > subscript2)
880	return(1);
881	else if (subscript1 < subscript2)
882	return(-1);
883	else
884	return(0);
885	}
886	
887	*str1_subscript = save_char1;
888	*str2_subscript = save_char2;
889	
890	return(strcmp(str1, str2));
891	}
892	#ifdef DEBUG
893	
894	static
895	print_delay_table()
896	{
897	u_long i;
898	
899	for(i=0; i<number_of_delays; ++i) {
900	fprintf(stdout, "str1 ", delay_table[i].composite_index);
901	if (delay_table[i].flag == NUMERIC_SUFFIX) {
902	fprintf(stdout, "%s(%d) ",
903	delay_table[i].tmax,
904	delay_table[i].numeric
905);
906	
907	fprintf(stdout, "delay from %s(%s) to %s(%s) = %d, %d",
908	STATE(delay_table[i].f_state),
909	delay_table[i].f_name,
910	STATE(delay_table[i].t_state),
911	delay_table[i].t_name,
912	delay_table[i].minimum,
913	delay_table[i].maximum
914);
915	fprintf(stdout, "\t(%0.1f, %0.1f, %0.1f)\n",
916	(float) delay_table[i].total,
917	(float) delay_table[i].squared_total,
918	(float) delay_table[i].number_of_elements
919);
920	}
921	#endif /* DEBUG */

Copyright 1989 Logic Modeling Systems		HEADER FILE include/HMBL.h	DATE 5/23/89 TIME 1:20:15 pm	PAGE # 1/1
LINE #	HEADER TEXT			
1	/* SCCS ID: HMBL.h rev 3.3, 5/9/89 at 15:30:29 */			
2	/* Header file for HMBL Analyzer */			
3	/* Success OK 0 */ /* okay exit status */			
4	/* Define FAIL 1 */ /* failure exit status */			
5	/* Define UNSHORT */ /* failure exit status */			
6	/* Define UNSHORT */ /* failure exit status */			
7	/* Define UNSHORT */ /* failure exit status */			
8	/* Define UNSHORT */ /* failure exit status */			
9	/* Define UNSHORT */ /* failure exit status */			
10	/* Define UNSHORT */ /* failure exit status */			
11	/* Define UNSHORT */ /* failure exit status */			
12	extern int Process_HMBL () /* called from main */			
13	extern char			
14	/* lm_malloc () /* malloc focus routine */			
15	/* lm_realloc () /* realloc focus routine */			
16	/* lm_get_input () /* returns input file name */			
17	extern unsigned			
18	/* lm_errors () /* returns number of errors */			
19	/* lm_get_lineno () /* returns current line number */			
20	/* lm_c_flag_on () /* returns state of c_flag */			
21	/* lm_d_flag_on () /* returns state of d_flag */			
22	/* lm_g_flag_on () /* returns state of g_flag */			
23	/* lm_i_flag_on () /* returns state of i_flag */			
24	/* lm_m_flag_on () /* returns state of m_flag */			
25	/* lm_p_flag_on () /* returns state of p_flag */			
26	/* lm_r_flag_on () /* returns state of r_flag */			
27	/* lm_s_flag_on () /* returns state of s_flag */			
28	/* lm_t_flag_on () /* returns state of t_flag */			
29	/* lm_v_flag_on () /* returns state of v_flag */			
30	extern void			
31	/* lm_free () /* free focus routine */			
32	/* lm_queue_message ()			
33	/* lm_queue_error ()			
34	/* lm_init_vars () /* initialize global variables */			
35	/* lm_set_c_flag () /* sets c_flag */			
36	/* lm_set_d_flag () /* sets d_flag */			
37	/* lm_set_g_flag () /* sets g_flag */			
38	/* lm_set_i_flag () /* sets i_flag */			
39	/* lm_set_m_flag () /* sets m_flag */			
40	/* lm_set_p_flag () /* sets p_flag */			
41	/* lm_set_r_flag () /* sets r_flag */			
42	/* lm_set_s_flag () /* sets s_flag */			
43	/* lm_set_t_flag () /* sets t_flag */			
44	/* lm_set_v_flag () /* sets v_flag */			
45	/* lm_set_input () /* sets input file name */			
46	/* lm_set_lineno () /* sets current line number */			
47	/* lm_error () /* reports counts errors */			
48	/* lm_warning () /* reports warnings */			
49	/* lm_syntax_error () /* reports a syntax error */			
50	/* lm_goto () /* longjmp()s back to the backend pass */			
51	/* end of header file for HMBL Analyzer */			
52				
53				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/Hconst.h	DATE 5/23/89 TIME 1:20:15 pm	PAGE # 1/2
LINE #	HEADER TEXT			
1	/* SOC3_ID: Soc3.h rev 3.1, 4/24/89 at 07:32:47 */			
2	/*			
3	Header File for Constraints for EMI Processor			
4	*/			
5	/* miscellaneous constants */			
6	#define BUF_SIZE 1024 /* character buffer size */			
7	#define ABS_MIN_PERIOD 1000 /* 1 GHz */			
8	#define ABS_MAX_PERIOD 1000000000 /* 1 KHz */			
9	#define MIN_PERIOD 40000 /* 25 MHz */			
10	#define MAX_PERIOD 6666667 /* 150 THz */			
11	#define UNDEFINED (-1000) /* undefined value */			
12	#define DEF_SETUP -1 /* for now */			
13	#define DEF_HOLD -1 /* for now */			
14	#define DEF_SAMPLE -1 /* for now */			
15	/* TTL voltage/current defaults */			
16	#define D_TTL_Vcc 5.0			
17	#define D_TTL_Vth 3.05			
18	#define D_TTL_Voh 2.9			
19	#define D_TTL_Vih 2.0			
20	#define D_TTL_Vil 0.8			
21	#define D_TTL_Val 0.4			
22	#define D_TTL_Vih 0.35			
23	#define D_TTL_Ioh 0.5			
24	#define D_TTL_Ial 2.0			
25	/* NMOS voltage/current defaults */			
26	#define D_NMOS_Vcc 5.0			
27	#define D_NMOS_Vth 3.2			
28	#define D_NMOS_Voh 2.9			
29	#define D_NMOS_Vih 2.0			
30	#define D_NMOS_Vil 0.8			
31	#define D_NMOS_Val 0.4			
32	#define D_NMOS_Vih 0.25			
33	#define D_NMOS_Ioh 0.5			
34	#define D_NMOS_Ial 0.5			
35	/* CMOS voltage/current defaults */			
36	#define D_CMOS_Vcc 5.0			
37	#define D_CMOS_Vth 4.7			
38	#define D_CMOS_Voh 4.5			
39	#define D_CMOS_Vih 3.5			
40	#define D_CMOS_Vil 1.0			
41	#define D_CMOS_Val 0.4			
42	#define D_CMOS_Vih 0.25			
43	#define D_CMOS_Ioh 0.5			
44	#define D_CMOS_Ial 0.5			
45	/* LCMOS voltage/current defaults */			
46	#define D_LCMOS_Vcc 3.3			
47	#define D_LCMOS_Vth 3.0			
48	#define D_LCMOS_Voh 2.8			
49	#define D_LCMOS_Vih 2.5			
50	#define D_LCMOS_Vil 0.6			
51	#define D_LCMOS_Val 0.4			
52	#define D_LCMOS_Vih 0.25			
53	#define D_LCMOS_Ioh 0.5			
54	#define D_LCMOS_Ial 0.5			
55	/* more miscellaneous constants */			
56	#define NOISE_MARGIN 0.4			
57	#define LCMOS_NOISE_MARGIN 0.3			
58	#define MAX_RISE /* maximum number of clock edges */			
59	#define MAX_FIRS 2560 /* maximum number of pins */			
60	#define MAX_SUBSCRIPT 65535 /* maximum subscript value */			
61	#define MAX_FB_SEQ_LEN 498 /* maximum feedback sequence length */			
62	#define MAX_RES_SEQ_LEN 65535 /* maximum reset sequence length */			
63	/* enumerated constants */			
64	#define ID_ATTR 0 /* used in attribute structure */			
65	#define CURRENT_SPEC 1			
66	#define CYCLE_SPEC 2			
67	#define set(T) syntact(textofnode(T)) /* abbreviation */			
68	#define ASSERT(x)			
69	if(!(x)) internal error: assertion failed: x)			
70	/* constant symbols used by the constraints */			
71	#define C_public (Q_lastreserved+1)			
72	#define C_private (Q_lastreserved+2)			
73	#define C_Hz (Q_lastreserved+3)			
74	#define C_Hz (Q_lastreserved+4)			
75	#define C_Hz (Q_lastreserved+5)			
76	#define C_Hz (Q_lastreserved+6)			
77	#define C_ms (Q_lastreserved+7)			
78	#define C_us (Q_lastreserved+8)			
79	#define C_ms (Q_lastreserved+9)			
80	#define C_ps (Q_lastreserved+10)			
81	#define C_infinity (Q_lastreserved+11)			
82	#define C_static (Q_lastreserved+12)			
83	#define C_internal (Q_lastreserved+13)			
84	#define C_ext1 (Q_lastreserved+14)			
85	#define C_ext2 (Q_lastreserved+15)			
86	#define C_TTL (Q_lastreserved+16)			
87	#define C_NMOS (Q_lastreserved+17)			
88	#define C_CMOS (Q_lastreserved+18)			
89	#define C_LCMOS (Q_lastreserved+19)			
90	#define C_custom (Q_lastreserved+20)			
91	#define C_Vcc (Q_lastreserved+21)			
92	#define C_Vth (Q_lastreserved+22)			
93	#define C_Vih (Q_lastreserved+23)			
94	#define C_Vil (Q_lastreserved+24)			
95	#define C_Val (Q_lastreserved+25)			
96	#define C_Ioh (Q_lastreserved+26)			
97	#define C_Ial (Q_lastreserved+27)			
98	#define C_Ioh (Q_lastreserved+28)			
99	#define C_Iol (Q_lastreserved+29)			
100	#define C_Iih (Q_lastreserved+30)			
101	#define C_Iil (Q_lastreserved+31)			
102	#define C_val (Q_lastreserved+32)			
103	#define C_store (Q_lastreserved+33)			
104	#define C_edge_rise (Q_lastreserved+34)			
105	#define C_edge_fall (Q_lastreserved+35)			
106	#define C_feedback (Q_lastreserved+36)			
107	#define C_loopalive (Q_lastreserved+37)			
108	#define C_in_sequence (Q_lastreserved+38)			
109	#define C_last_cycle (Q_lastreserved+39)			
110	#define C_hard (Q_lastreserved+40)			
111	#define C_medium (Q_lastreserved+41)			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/Hconst.h	DATE 5/23/89	PAGE # 2/3
LINE #		HEADER TEXT		
121		# define C_hard_medium (Q_lastreserved+42)		
122		# define C_rise (Q_lastreserved+43)		
123		# define C_fall (Q_lastreserved+44)		
124		# define C_low (Q_lastreserved+45)		
125		# define C_high (Q_lastreserved+46)		
126		# define C_float (Q_lastreserved+47)		
127		# define C_valid (Q_lastreserved+48)		
128		# define C_arry (Q_lastreserved+49)		
129		# define C_min (Q_lastreserved+50)		
130		# define C_typ (Q_lastreserved+51)		
131		# define C_max (Q_lastreserved+52)		
132		# define C_on (Q_lastreserved+53)		
133		# define C_off (Q_lastreserved+54)		
134		# define C_missing_delays (Q_lastreserved+55)		
135		# define C_io_store_changes (Q_lastreserved+56)		
136		# define C_trace_delay (Q_lastreserved+57)		
137		# define C_pull_up (Q_lastreserved+58)		
138		# define C_pull_down (Q_lastreserved+59)		
139		struct dev /* constraint's device structure */		
140		{		
141		short		
142		edges, /* number of clock edges */		
143		num_in, /* number of input pins */		
144		num_out, /* number of output pins */		
145		num_io, /* number of I/O pins */		
146		num_power, /* number of power pins */		
147		num_ground, /* number of ground pins */		
148		num_ac, /* number of AC pins */		
149		pin_out, /* pin counter */		
150		spin_cnt, /* adapter pin counter */		
151		reset_cnt, /* reset sequence counter */		
152		num_seqs, /* number of sequences */		
153		fb, /* is feedback same time? */		
154		pre_seq_length, /* length of prefeedback sequences */		
155		fb_seq_length, /* length of feedback sequences */		
156		post_seq_length, /* length of postfeedback sequences */		
157		ultra_fast, /* ultra fast mode involved */		
158		dis_tin_check, /* disable timing checking */		
159		inh_tin_measure, /* inhibit timing measurement */		
160		delays_seen, /* are there any delays? */		
161		use_default, /* has default delay been seen? */		
162		has_store_pins, /* set to indicate device has eval/store pins */		
163		has_adapter_map, /* set to indicate adapter map has been set */		
164		has_package_map, /* set to indicate package map has been set */		
165		has_feedback_pins, /* set to indicate device has feedback pins */		
166		unassigned logic		
167		clk_period1, /* first clock period in picoseconds */		
168		clk_period2, /* second clock period in picoseconds */		
169		min_default, /* minimum default delay */		
170		typ_default, /* typical default delay */		
171		max_default, /* maximum default delay */		
172		tech_tree, /* technology tree */		
173		symbol		
174		dev_name, /* device name */		
175		dev_type, /* device type */		
176		mod_name, /* module name */		
177		clk_type, /* clock type */		
178		technology, /* technology */		
179		missing_delays, /* missing delays */		
180		io_store_changes, /* I/O store changes */		
181		double		
182		def_hold, /* default hold time in picoseconds */		
183		def_setup, /* default setup time in picoseconds */		
184		def_sample, /* default sample time in picoseconds */		
185		Vcc, /* DC Vcc */		
186		Vthh, /* high threshold voltage */		
187		Vthl, /* low threshold voltage */		
188		Vih, /* logical high voltage */		
189		Vil, /* logical low voltage */		
190		Vsh, /* soft drive high voltage */		
191		Val, /* soft drive low voltage */		
192		Ioh, /* output high current in milliamperes */		
193		Iol, /* output low current in milliamperes */		
194		Iih, /* input high current in milliamperes */		
195		Iil, /* input low current in milliamperes */		
196		Ial, /* soft low current in milliamperes */		
197		Iah, /* soft high current in milliamperes */		
198		struct pin_attr /* pin attributes */		
199		{ char use_flag, /* ID_ATTR, CURRENT_SPEC, CYCLE_SPEC */		
200		union		
201		{		
202		symbol id, /* if ID_ATTR */		
203		struct		
204		{		
205		symbol cur, /* current specifier: if(0)[1h] pull (up/down) */		
206		double val, /* current value in milliamperes (only if pull) */		
207		} cur_spec, /* if CURRENT_SPEC */		
208		struct		
209		{		
210		symbol seq, /* sequence specifier */		
211		drv, /* drive specifier */		
212		} cyc_spec, /* if CYCLE_SPEC */		
213		} attr,		
214		} data format		
215		struct pins /* pin structure */		
216		{		
217		symbol name, /* pin name */		
218		symbol pkg_name, /* package pin name */		
219		symbol alias, /* simulator pin name */		
220		ushort number, /* adapter pin number */		
221		ushort trace_delay, /* trace delay to this pin in ps */		
222		ushort num_attr, /* number of attributes */		
223		struct pin_attr *attr, /* pointer to array of attributes */		
224		struct timing		
225		{		
226		symbol		
227		major_state, /* pin state for this pin */		
228		minor_state, /* other pin state involved */		
229		minor_pin_name, /* other pin involved */		
230		unassigned logic		
231		minimum, /* minimum time in picoseconds */		
232		typical, /* typical time in picoseconds */		
233		maximum, /* maximum time in picoseconds */		
234		struct timing *next, /* pointer to next timing spec */		
235		} timing_list, /* pointer to linked list of timing specs */		
236		} pins,		
237		} sequence /* sequence structure */		
238		{		
239		symbol pin_name, /* name of pin */		
240		tree bits, /* tree representing sequence */		
241		ushort fb_kid, /* which subtree has the feedback subsequence */		
242		} sequences, /* pointer to array of sequence structures */		
243		}		
244		}		
245		extern void		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/Hconst.h	DATE 5/23/89 TIME 1:20:15 pm	PAGE # 3/4
LINE #	HEADER TEXT			
241	in_print_message () /* to print without stdio */			
242	error () /* reports a constraint error */			
243	constraint () /* reports a constraint warning */			
244	tr_entry () /* traces entry to a subtree */			
245	tr_exit () /* traces exit from a subtree */			
246	add_attr () /* adds an attribute to a pin */			
247	buf_instead () /* buffers up text for error reporting */			
248	buf_string () /* buffers up the text of a string token */			
249	check_state () /* makes sure passed state is legal */			
250	process_timing () /* timing specification processor */			
251	walk_1 () /* first walk of the constraint */			
252	walk_2 () /* second walk of the constraint */			
253	walk_3 () /* third walk of the constraint */			
254	walk_4 () /* fourth walk of the constraint */			
255	extern ushort			
256	compare () /* compares symbols without regard to case */			
257	is_store_or_oval () /* returns 1 iff passed pin is a store or oval pin */			
258	do_pin_name () /* processes pin names in walk_1 */			
259	extern struct dev *the /* the global device structure pointer */			
260	extern char buf [BUF_SIZE] /* the global character buffer */			
261	extern symbol gen_name () /* generates a name gives a symbol and a number */			
262	extern symbol prepend_underscore () /* prepends _ to the passed symbol */			
263	extern double check_num () /* checks a passed number */			
264	extern unsigned long check_int () /* makes sure number is an integer */			
265				
266	/* end of Header File for Constraint for HBL Processor */			
267				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/Hparse.h	DATE 5/23/89	PAGE # 1/5
LINE #		HEADER TEXT		
1		/* SCCS ID: Hparse.h rev 3.1.1, 4/24/89 at 07:32:50 */		
2		#		
3		# define END_OF_FILE 1		
4		# define P_ID 2		
5		# define P_NUM 3		
6		# define P_STR 4		
7		# define P_NAME 5		
8		# define M_FAST 6		
9		# define M_USAGE 7		
10		# define M_REPORT 8		
11		# define M_TECHNOLOGY 9		
12		# define M_CLOCK_TYPE 10		
13		# define M_DIS_TIM_CHECK 11		
14		# define M_TIM_TIM_MEASURE 12		
15		# define M_HOLD_TIME 13		
16		# define M_SETUP_TIME 14		
17		# define M_SAMPLE_TIME 15		
18		# define M_SPEED 16		
19		# define M_TECHNOLOGY 17		
20		# define M_INITIALIZE 18		
21		# define M_IN_FINS 19		
22		# define M_IO_FINS 20		
23		# define M_MC_FINS 21		
24		# define M_COT_FINS 22		
25		# define M_POWER_FINS 23		
26		# define M_GROUND_FINS 24		
27		# define M_DELAY 25		
28		# define M_DEFAULT_DELAY 26		
29		# define M_PACKAGE 27		
30		# define M_ADAPTER 28		
31		# define M_PIN_MAP 29		
32		# define M_UNIT 30		
33		# define M_TEN_SPEC 31		
34		# define M_NEGATIVE 32		
35		# define M_INIT_SEQ 33		
36		# define M_LIST 34		
37		# define M_REPEAT 35		
38		# define M_FEEDBACK 36		
39		# define M_PIN_CORR 37		
40		# define M_PIN_NAME 38		
41		# define M_NAMES 39		
42		# define M_ATTN 40		
43		# define M_FROM 41		
44		# define M_TO 42		
45		# define M_PIN_STATE 43		
46		# define M_TIRING 44		
47		# define M_SPEC 45		
48		# define M_MAPPING 46		
49		# define Q_lastresname 46		
50		# define Q_firstresname 47		
51		# define T_device_name 47 /* device_name */		
52		# define T_ultra_fast 48 /* ultra_fast */		
53		# define T_device_usage 49 /* device_usage */		
54		# define T_report 50 /* report */		
55		# define T_modeler_name 51 /* modeler_name */		
56		# define T_clock_type 52 /* clock_type */		
57		# define T_disable_timing_checking 53 /* disable_timing_checking */		
58		# define T_inhibit_timing_measurement 54 /* inhibit_timing_measurement */		
59		# define T_device_hold_time 55 /* device_hold_time */		
60		# define T_device_setup_time 56 /* device_setup_time */		
61		# define T_device_sample_time 57 /* device_sample_time */		
62		# define T_device_speed 58 /* device_speed */		
63		# define T_technology 59 /* technology */		
64		# define T_initialize 60 /* initialize */		
65		# define T_in_pin 61 /* in_pin */		
66		# define T_io_pin 62 /* io_pin */		
67		# define T_mc_pin 63 /* mc_pin */		
68		# define T_out_pin 64 /* out_pin */		
69		# define T_power_pin 65 /* power_pin */		
70		# define T_ground_pin 66 /* ground_pin */		
71		# define T_delay 67 /* delay */		
72		# define T_default_delay 68 /* default_delay */		
73		# define T_package_mapping 69 /* package_mapping */		
74		# define T_adapter_mapping 70 /* adapter_mapping */		
75		# define T_pin_name_mapping 71 /* pin_name_mapping */		
76		# define T_da 72 /* da */		
77		# define T_ch 73 /* ch */		
78		# define T_eq 74 /* eq */		
79		# define T_lp 75 /* lp */		
80		# define T_rp 76 /* rp */		
81		# define T_is 77 /* is */		
82		# define T_zs 78 /* zs */		
83		# define T_co 79 /* co */		
84		# define T_from 80 /* from */		
85		# define T_to 81 /* to */		
86		# define Q_lastresname 81		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/arp.h	DATE 5/23/89 TIME 1:20:16 pm	PAGE # 1/6
LINE #	HEADER TEXT			
1	/* SCCS ID: arp.h rev 3.1, 4/24/89 at 07:32:52 */			
2	/*			
3	This file contains the ARP request and response			
4	constants and data structures. For information			
5	on ARP and RARP see the DCM protocol handbook,			
6	volume 3, pp. 3-615.			
7	*/			
8	/*			
9	ARP op-codes */			
10	/*			
11	#define ARP_REQUEST 1			
12	#define ARP_REPLY 2			
13	/*			
14	ARP hardware types */			
15	/*			
16	#define ARP_ETHERNET 1			
17	#define ARP_IP_ETHERNET 2			
18	#define ARP_RADIO_X25 3			
19	#define ARP_TOKEN_RING 4			
20	#define ARP_CHAOS 5			
21	/*			
22	ARP protocols types */			
23	/*			
24	#define ARP_IP 0x0800			
25	#define ARP_ARP 0x0806			
26	#define ARP_IPX 0x0807			
27	#define ARP_RARP 0x8035			
28	#define ARP_LOOP 0x9000			
29	/*			
30	ARP address sizes */			
31	/*			
32	#define ARP_HARDWARE_SIZE 6			
33	#define ARP_PROTOCOL_SIZE 4			
34	/*			
35	special ethernet addresses */			
36	/*			
37	#define BROADCAST "\377\377\377\377\377\377"			
38	#define NULL_ADDRESS "\000\000\000\000\000\000"			
39	/*			
40	ARP data packet structure */			
41	/*			
42	struct arp_packet {			
43	unsigned short arp_hardware_type;			
44	unsigned short arp_protocol_type;			
45	unsigned char arp_hardware_address_length;			
46	unsigned char arp_protocol_address_length;			
47	unsigned short arp_opcode;			
48	unsigned char arp_source_hardware_address[ARP_HARDWARE_SIZE];			
49	unsigned long arp_source_protocol_address;			
50	unsigned char arp_target_hardware_address[ARP_HARDWARE_SIZE];			
51	unsigned long arp_target_protocol_address;			
52	};			
53	/*			
54	ethernet packet structure */			
55	/*			
56	struct ethernet_arp_packet {			
57	unsigned char destination[6];			
58	unsigned char source[6];			
59	unsigned short type;			
60	struct arp_packet			
61	};			
62	/*			
63	typedef struct icmp_header			
64	{			
65	u_char type;			
66	u_char code;			
67	u_short checksum;			
68	u_short identifier;			
69	u_short sequence_no;			
70	}ICMP_HEADER;			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/bus.h	5/23/89	1/7
			TIME	1:20:16 pm
LINE #		HEADER TEXT		
1		/* SOCS_ID: bus.h rev 3.1.1, 4/24/89 at 07:32:54 */		
2		/*		
3		/* The bus error handler		
4		/* see, s4020 near manual, page 6		
5		/*		
6		/*		
7		/* we are only interested in errors for bus cycles		
8		/*		
9		typedef struct		
10		{		
11		u_short ar;		
12		u_long pc;		
13		u_short vector;		
14		u_short internal_reg;		
15		u_short spec_status_word;		
16		u_short internal_pipe_c;		
17		u_short internal_pipe_b;		
18		u_char address;		
19		u_long internal_reg2;		
20		u_long data_out;		
21		u_long internal_reg3;		
22		u_long internal_reg4;		
23		u_long data_in;		
24		} STACK_FRAME;		
25				
26				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/common.h	DATE 5/23/89 TIME 1:20:17 pm	PAGE # 1/8
LINE #	HEADER TEXT			
1	/* SCCS ID: common.h rev 3.1, 4/24/89 at 07:32:57 */			
2				
3	#ifndef TYPES			
4	#define TYPES			
5	#typedef unsigned char u_char;			
6	#typedef unsigned long u_long;			
7	#typedef unsigned short u_short;			
8	#endif			
9				
10	#ifndef NULL			
11	#define NULL 0			
12	#endif			
13				
14	#define TRUE 1			
15	#define FALSE 0			
16				
17	#define SUCCESS 1			
18	#define FAILURE 0			
19				
20	#define NOT_SUCCESS_OR_FAILURE 2			
21				
22	#ifndef VMS			
23	#define EXIT_NO_ERROR 1			
24	#define EXIT_FATAL_ERROR 0			
25	#else			
26	#define EXIT_NO_ERROR 0			
27	#define EXIT_FATAL_ERROR 1			
28	#endif			
29				
30				
31	/*			
32	* Defined to keep from having to use string.h and/or strings.h			
33	* Basically, a BSD/SYS5 portability consideration.			
34	*/			
35	extern char *strcat();			
36	extern char *strncat();			
37	extern char *strcpy();			
38	extern char *strncpy();			
39				
40	extern char *strchr();			
41	extern char *strrchr();			
42	extern char *strpbrk();			
43	extern char *strtok();			
44				
45	extern int strcmp();			
46	extern int strncmp();			
47	extern int strcmpx();			
48	extern int strcmps();			
49	extern int strcmpsp();			
50				
51	/*			
52	* Defined to keep from having to use malloc.h and/or memory.h			
53	* Basically, a BSD/SYS5 portability consideration.			
54	*/			
55	#ifndef VMS /* a VMS portability consideration */			
56	extern char *malloc();			
57	extern char *realloc();			
58	#endif			
59				
60	#ifndef MODELER			
61				
62	/*			
63	* The reason for defining sprintf() here, only for the modeler.			
64	* is that on some machines (most notably SYS5) sprintf() returns			
65	* an integer. Therefore, the definition of sprintf() must be			
66	* picked up in from <stdio.h> but <stdio.h> can not be used as			
67	* the modeler the concept of <stdio.h> does not exist.			
68	*/			
69	extern char *sprintf();			
70	#endif			
71				
72	/*			
73	* Well, it turns out that Sun systems must also have sprintf() defined.			
74	*/			
75	#ifndef SUN			
76	extern char *sprintf();			
77	#endif			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/cpu.h	DATE 5/23/89 TIME 1:20:17 pm	PAGE # 1/9
HEADER TEXT				
1	/* SOC3_ID: cpu.h rev 3.3, 5/9/89 at 17:45:44 */			
2	#define CPU_RAM 0x00000000 /* DRAM start */			
3	#define CPU_BOOT_EPRON 0x00000000 /* EPRON start during boot */			
4	#define CPU_BOOT_RAM 0x00100000 /* DRAM start during boot */			
5	#define CPU_RAM_SIZE 0x00400000 /* DRAM size (4MB) */			
6	#define CPU_VME24_RAM 0x00400000 /* VME DRAM start */			
7	#define CPU_VME24_PERIPH 0x10000000 /* VME24 peripherals start */			
8	#define CPU_VME24_PERIPH_SIZE 0x00400000 /* (4MB) */			
9	#define CPU_NVRAM 0x10000000 /* Non-volatile SRAM start */			
10	#define CPU_NVRAM_SIZE 0x00020000 /* (2KB/2K neg) */			
11	#define CPU_TIMER 0x10010000 /* 8254 start (16M/16 reg) */			
12	#define CPU_TIMER_SIZE 0x10010000 /* 8254 start (16M/16 reg) */			
13	#define CPU_LANCE_DATA_REG 0x10c40002 /* 82590 data reg (word) */			
14	#define CPU_LANCE_ADDR_REG 0x10c40006 /* 82590 address reg (word) */			
15	#define CPU_ID_PROG 0x10c30000 /* ID PROM start */			
16	#define CPU_ID_PROG_SIZE 0x00000020 /* (128K/32 reg) */			
17	#define CPU_CTRL_REG 0x10c60000 /* Control register (4B) */			
18	#define CPU_CLEAR_ERRORS_REG 0x10c60000 /* Byte write to clear errors */			
19				
20	#define CPU_READ_CONTROL 0x10c60000 /* Read-only control byte */			
21	#define CPU_FUNC_CONTROL 0x10c60001 /* Read/write control byte */			
22	#define CPU_INTR_CONTROL 0x10c60002 /* Read/write control byte */			
23	#define CPU_GLOBAL_INTR_ENA 0x80 /* Enable CPU_INTR_CONTROL */			
24	#define CPU_MISC_CONTROL 0x10c60003 /* Read/write control byte */			
25				
26	#define CPU_PAR_ERR_REG 0x10c70000 /* Parity error status/addr */			
27				
28	#define CPU_EPRON 0x10d00000 /* EPRON start, always */			
29	#define CPU_LANE_A 0x00000000 /* TNC and backplane */			
30	#define CPU_LANE_B 0x00000000 /* TNC and backplane */			
31	#define CPU_LANE_C 0x00000000 /* TNC and backplane */			
32	#define CPU_LANE_D 0x00000000 /* TNC and backplane */			
33	#define CPU_VME23_PERIPH 0x10000000 /* VME23 peripherals start */			
34	#define CPU_VME23_PERIPH_SIZE 0x10000000 /* (256MB) */			
35				
36	typedef struct			
37	{			
38	unsigned			
39	{			
40	not_68020_master:1, /* Negated if 68020 master when parity error */			
41	not_vme_master:1, /* Negated if VME master when parity error */			
42	not_lance_master:1, /* Negated if LANCE master when parity error */			
43	not_error_hi:1, /* Negated if parity error on high byte */			
44	not_error_mi:1, /* Same for upper middle byte */			
45	not_error_lo:1, /* Same for lower middle byte */			
46	not_error_lo:1, /* Same for low byte */			
47	error_addr:22, /* Address of parity error (bits 21:0) */			
48	}			
49	} cpu_err_reg;			
50				
51	/*			
52	* Accesses to the control register should only be made using the following			
53	* structure, whenever during diagnostics. In particular, the CPU_READ_CONTROL			
54	* read-only byte lives a dual life as the CPU_CLEAR_ERRORS write-only byte,			
55	* so that a longword read-modify-write cycle on the control register			
56	* is possible.			
57	*/			
58	typedef struct			
59	{			
60	/*			
61	* CPU_READ_CONTROL byte, read-only control register.			
62	* Do not, under any circumstances, write any of these bits-- this is			
63	* at the same address as the write-only CPU_CLEAR_ERRORS location.			
64	*/			
65	unsigned			
66	{			
67	lance_busy:1, /* LANCE can't be accessed */			
68	not_parity_intr:1, /* No parity error/interrupt pending */			
69	not_dtr:1, /* DTR part of DTR is asserted */			
70	debug_mode:1, /* VME24 can't access enhanced SRAM */			
71	dipswitch:4, /* 4-position dipswitch, 0=on */			
72	}			
73	/*			
74	* CPU_FUNC_CONTROL byte, read/write control register.			
75	/*			
76	* not_parity_value:1, /* Inverse of parity bit written during */			
77	* parity_force:1, /* parity testing */			
78	* enable_parity:1, /* Enable parity testing */			
79	* enable_non_volatile_srwm:1, /* Enable non-volatile SRAM writes */			
80	* enable_lance:1, /* Allow the LANCE to become master */			
81	* enable_cache:1, /* Enable the 68020's 1-cache */			
82	* timer_gate_0:1, /* 8254 timer gate 0 */			
83	* timer_gate_1:1, /* 8254 timer gate 1 */			
84	/*			
85	* CPU_INTR_CONTROL byte, read/write control register.			
86	/*			
87	* global_intr_ena:1, /* Enable ALL interrupts, even NMI */			
88	* parity_intr_ena:1, /* Enable parity interrupt */			
89	* timer_intr_ena_0:1, /* Enable timer channel 0 interrupt */			
90	* timer_intr_ena_1:1, /* Enable timer channel 1 interrupt */			
91	* timer_intr_ena_2:1, /* Enable timer channel 2 interrupt */			
92	* vme_intr_ena:1, /* Enable interrupt from VME */			
93	* doorbell:1, /* Doorbell interrupt to 68020 */			
94	/*			
95	* CPU_MISC_CONTROL byte, mostly read/write control register.			
96	/*			
97	* not_load_led:1, /* Negate bit to turn on orange LED */			
98	* not_test_led:1, /* Negate bit to turn on yellow LED */			
99	* fault_led:1, /* Turn on red LED */			
100	* control_spare:1, /* Unassigned bit */			
101	* not_lance_reset:1, /* Negate bit to reset LANCE (UNIMPLEMENTED) */			
102	* not_tng_intr:1, /* Actual pollable interrupt from TNC */			
103	* tng_test_data:1, /* Test with special TNC interface fixture */			
104	* suicide:1, /* Software hard reset */			
105	/*			
106	} cpu_control_reg_struct;			
107				
108	/*			
109	* Useful macros.			
110	/*			
111	#define CPU_ENABLE_INTERRUPTS \			
112	(unsigned char) CPU_INTR_CONTROL = CPU_GLOBAL_INTR_ENA			
113	#define CPU_DISABLE_INTERRUPTS \			
114	(unsigned char) CPU_INTR_CONTROL &= ~CPU_GLOBAL_INTR_ENA			
115	/*			
116	/*			
117	/*			
118	* Dipswitch stuff.			
119	/*			
120	#define CPU_SW_UP 1			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/cpu.h	5/23/89	
			TIME	2/10
			1:20:17 pm	
LINE #	HEADER TEXT			
121	#define CPU_SW_DOWN 0			
122	/*			
123	* Call cpu_sw(s-bit dipswitch register, number between 1 and 4)			
124	* Returns value of either CPU_SW_UP or CPU_SW_DOWN			
125	*/			
126	#define cpu_sw(reg, swpos) (((reg) >> (4 - (swpos))) & 0x1)			
127	/*			
128	* LED constants:			
129	*/			
130	#define LED_ON 1			
131	#define LED_OFF 0			
132	#define not_LED_ON 0			
133	#define not_LED_OFF 1			
134	/*			
135	* cpu_clear_errors() macro			
136	*/			
137	#define cpu_clear_errors() *((u_char *)CPU_CLEAR_ERRORS_REG)= 0			
138	/*			
139	* For software delays of the form:			
140	* register u_long delay_count;			
141	* delay_count= CONSTANT;			
142	* while (--delay_count)			
143	* CPU_LOOPS_PER_5_MICROSECONDS is the scale factor for computing CONSTANT.			
144	* With a value of 10, each loop takes 500ns.			
145	*/			
146	#define CPU_LOOPS_PER_5_MICROSECONDS 10			
147	/*			
148	* Macros to delay for a given number of microseconds or milliseconds			
149	*/			
150	#define delay_microseconds(v, n) \			
151	{ v= (CPU_LOOPS_PER_5_MICROSECONDS * n) / 5; \			
152	while (--v); }			
153	#define delay_milliseconds(v, n) \			
154	{ v= 100 * CPU_LOOPS_PER_5_MICROSECONDS * n; \			
155	while (--v); }			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/cpu_diag.h	5/23/89	1/11
			TIME	1:20:17 pm
HEADER TEXT				
LINE #				
1	/* SCCS_ID: cpu_diag.h, rev. 1.1.1, 4/24/89, at 07:33:01 */			
2	/*			
3	* cpu_diag.h			
4	* Copyright 1989 Logic Modeling Systems, Inc.			
5	* map			
6	*/			
7	* Definitions for general use in CPU diagnostics:			
8	*/			
9				
10	#define CPUDIAG_NO_ERROR 0 /* No errors found */			
11	#define CPUDIAG_ERROR_DRAM 1 /* One blink */			
12	#define CPUDIAG_ERROR_IDPRON 2 /* Two blinks */			
13	#define CPUDIAG_ERROR_EPRON 3 /* Three blinks */			
14	#define CPUDIAG_ERROR_EXCEPTION 4 /* Four blinks */			
15	#define CPUDIAG_ERROR_STATUS 5 /* Five blinks */			
16				
17	#define INLINE_ROOT asm("bral _bop_start")			
18				
19	#define INLINE_HYPERLINK asm("movl \$0x04000, a0"); asm("bral _hl_rom")			
20				
21	#define INPUT_BUF_LENGTH 20 /* maximum input line length */			
22				
23	#define SIZE_BYTE 1			
24	#define SIZE_WORD 2			
25	#define SIZE_LONG 4			
26				
27	#define INPUT_BAD 0			
28	#define INPUT_GOOD 1			
29	#define INPUT_ZERO 2			
30	#define INPUT_ABORT 3			
31				
32	#define CPUDIAG_RAM_START 0x00001000			
33	#define CPUDIAG_RAM_END 0x00100000			
34				
35	#define CPUNON_ADDR_PROMPT ": "			
36	#define CPUNON_DATA_PROMPT "> "			
37				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/cpu_frame.h	DATE 5/23/89	PAGE # 1/12
			TIME 1:20:17 pm	

LINE #	HEADER TEXT
1	/* SOC3_ID: cpu_frame.h rev 3.1, 4/24/89 at 07:33:04 */
2	/*
3	* cpu_frame.h
4	* Definitions for 68010 CPU stack frames
5	*/
6	/*
7	* Stack frame types:
8	*/
9	#define STACK_FRAME_GENERIC 0
10	#define STACK_FRAME_THERMANY 1
11	#define STACK_FRAME_SIXWORD 2
12	#define STACK_FRAME_CP_MIDINST 3
13	#define STACK_FRAME_BUS_SHORT 10
14	#define STACK_FRAME_BUS_LONG 11
15	#define STACK_FRAME_RERUN 1
16	#define STACK_FRAME_DONT_RERUN 0
17	#define STACK_FRAME_SIZE_BYTE 1
18	#define STACK_FRAME_SIZE_WORD 2
19	#define STACK_FRAME_SIZE_TRIPLE 3
20	#define STACK_FRAME_SIZE_LONG 0
21	/*
22	* Generic exception stack frame.
23	* This structure is common to all stack frames, and suffices for interrupts,
24	* format errors, TRAPs, illegal instructions, A-line and P-line privilege
25	* violations, and coprocessor pre-instruction exceptions:
26	*/
27	typedef struct
28	{
29	u_short ar;
30	u_long pc;
31	unsigned type:4,
32	vec:12,
33	} stack_frame_generic;
34	/*
35	* Six-word exception stack frame.
36	* This is used for CHK, CMT2, opTRAPs, TRAPs, TRAPV, trace, zero divide,
37	* and coprocessor post-instruction exceptions.
38	*/
39	typedef struct
40	{
41	u_short ar;
42	u_long pc;
43	unsigned type:4,
44	vec:12,
45	u_long inst_addr;
46	} stack_frame_sixword;
47	/*
48	* Coprocessor mid-instruction stack frame.
49	* This should never happen in this system.
50	*/
51	typedef struct
52	{
53	u_short ar;
54	u_long pc;
55	unsigned type:4,
56	vec:12,
57	u_long inst_addr;
58	u_long instnall;
59	u_long instnall2;
60	} stack_frame_cp_midinst;
61	/*
62	* Short bus error stack frame:
63	*/
64	typedef struct
65	{
66	u_short ar;
67	u_long pc;
68	unsigned type:4,
69	vec:12,
70	u_short instnall;
71	unsigned fault_c:1,
72	fault_b:1,
73	rarus_c:1,
74	rarus_b:1,
75	bogus:3,
76	rarus_flag:1,
77	rnc:1,
78	read:1,
79	size:2,
80	bogus2:1,
81	fc:3,
82	u_short pipe_c;
83	u_short pipe_b;
84	u_long fault_addr;
85	u_long instnall2;
86	u_long data_out;
87	u_long instnall;
88	} stack_frame_bus_short;
89	/*
90	* Long bus error stack frame:
91	*/
92	typedef struct
93	{
94	u_short ar;
95	u_long pc;
96	unsigned type:4,
97	vec:12,
98	u_short instnall;
99	unsigned fault_c:1,
100	fault_b:1,
101	rarus_c:1,
102	rarus_b:1,
103	bogus:3,
104	rarus_flag:1,
105	rnc:1,
106	read:1,
107	size:2,
108	bogus2:1,
109	fc:3,
110	u_short pipe_c;
111	u_short pipe_b;
112	u_long fault_addr;
113	u_long instnall2;
114	u_long data_out;
115	u_long instnall;
116	} stack_frame_bus_long;

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/cpu_frame.h	5/23/89	2/13
			TIME 1:20:17 pm	
LINE #	HEADER TEXT			
121	u_short pipe_b;			
122	u_long fault_addr;			
123	u_long internal1;			
124	u_long data_out;			
125	u_long b_addr;			
126	u_long internal1;			
127	u_long data_in;			
128	u_long internal4[11];			
129) stack_frame_bus_long;			
130				
131	typedef struct			
132	{			
133	u_long fault_addr;			
134	u_char print_flag;			
135	u_char rerun_flag;			
136	u_char occur_flag;			
137	u_char size;			
138	u_char rxc;			
139	u_char read;			
140	u_char rc;			
141) bus_error_control_struct;			
142				
143	#define frame_set_trace_mode(ar, ctl) \			
144	ar = (ar & (0x1fff)) + (ctl << 14)			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/cpu_vec.h	DATE 5/23/89	PAGE # 1/14
LINE #		HEADER TEXT		
1		/* SOCS ID: cpa_vec.h rev 3.1, 4/24/89 at 07:33:06 */		
2		/*		
3		* cpu_vec.h		
4		* Definitions relating to CPU exception vectors		
5		*/		
6		/*		
7		* General definitions		
8		*/		
9		#define VEC_BASE_ADDRESS 0x00000000		
10		#define VEC_END_ADDRESS 0x00000400		
11		#define VEC_NUM_VECTS 256		
12		/*		
13		* Define exception indices for miscellaneous exceptions		
14		*/		
15		#define VEC_RESET_ISR 0		
16		#define VEC_RESET_PC 1		
17		#define VEC_BUS_ERROR 2		
18		#define VEC_ADDRESS_ERROR 3		
19		#define VEC_ILLEGAL_INSTRUCTION 4		
20		#define VEC_ZERO_DIVIDE 5		
21		#define VEC_CHK_BOUNDS 6		
22		#define VEC_CONDITIONAL_TRAP 7		
23		#define VEC_PRIVILEGE_VIOLATION 8		
24		#define VEC_TRACE 9		
25		#define VEC_LDR_1010_EMULATOR 10		
26		#define VEC_LDR_1111_EMULATOR 11		
27		#define VEC_RESERVED_12 12		
28		#define VEC_CP_PROTOCOL_VIOL 13		
29		#define VEC_F0R4Y_ERROR 14		
30		#define VEC_UNINITIALIZED_INTN 15		
31		#define VEC_RESERVED_16 16		
32		#define VEC_RESERVED_17 17		
33		#define VEC_RESERVED_18 18		
34		#define VEC_RESERVED_19 19		
35		#define VEC_RESERVED_20 20		
36		#define VEC_RESERVED_21 21		
37		#define VEC_RESERVED_22 22		
38		#define VEC_RESERVED_23 23		
39		/*		
40		* Interrupts (operations and the auto-vectors)		
41		*/		
42		#define VEC_SPURIOUS_INTERRUPT 24		
43		#define VEC_LEVEL_1_INTERRUPT 25		
44		#define VEC_CB_INTERRUPT 26		
45		#define VEC_LEVEL_2_INTERRUPT 27		
46		#define VEC_VME_INTERRUPT 28		
47		#define VEC_TIMER_INTERRUPT 29		
48		#define VEC_DQART_INTERRUPT 30		
49		#define VEC_LANCE_DOORBELL_INTR 31		
50		#define VEC_PARITY_INTERRUPT 31		
51		#define VEC_LEVEL_7_INTERRUPT 31		
52		/*		
53		* Traps		
54		*/		
55		#define VEC_TRAP_0 32		
56		#define VEC_TRAP_VHTX 32		
57		#define VEC_TRAP_1 33		
58		#define VEC_TRAP_RTSCOPE 33		
59		#define VEC_TRAP_2 34		
60		#define VEC_TRAP_3 35		
61		#define VEC_TRAP_4 36		
62		#define VEC_TRAP_5 37		
63		#define VEC_TRAP_6 38		
64		#define VEC_TRAP_7 39		
65		#define VEC_TRAP_8 40		
66		#define VEC_TRAP_9 41		
67		#define VEC_TRAP_10 42		
68		#define VEC_TRAP_11 43		
69		#define VEC_TRAP_12 44		
70		#define VEC_TRAP_13 45		
71		#define VEC_TRAP_14 46		
72		#define VEC_TRAP_15 47		
73		/*		
74		* Vectors 48 through 51 are coprocessor exceptions. Since we don't		
75		* implement any coprocessors, we don't particularly care a whole lot		
76		* about these exceptions.		
77		* Vectors 52 through 55 are unassigned and reserved.		
78		*/		
79		/*		
80		* Note that the user-defined vectors are only useful in the event of		
81		* an external vector-generating interrupt source. This can happen		
82		* for VMEbus interrupts (if there were any VMEbus devices) or if the		
83		* EPROMs are burned wrong or go bad and a wrong "auto-vector" is		
84		* fetched during interrupt acknowledge EPROM read.		
85		*/		
86		#define VEC_USER_DEFINED_VECTOR 64		
87		#define VEC_LAST_USER_DEFINED 255		
88		/*		
89		* vec_handler()		
90		* This macro sets up the appropriate exception vector to point to the		
91		* designated handler function.		
92		*/		
93		#define vec_handler(vector, handler) \		
94		* (int *) (VEC_BASE_ADDRESS + (vector * 4)) = (int) handler		
95		/*		
96		* vec_entry_point()		
97		* Defines an alternate entry point for a function (especially an exception		
98		* handler). The argument should be preceded by an underline character.		
99		*/		
100		#define vec_entry_point(label) \		
101		* asm(".globl label"); asm("label:");		
102		/*		
103		* vec_handler_start()		
104		* Saves all registers (except stack pointer).		
105		* Should be used after vec_entry_point() in exception handlers.		
106		*/		
107		#define vec_handler_start() \		
108		* asm("movl \$0xFFFF, sp@-");		
109		/*		
110		*		

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/cpu_vec.h	5/23/89	
			TIME	1:20:17 pm
				2/15
LINE #	HEADER TEXT			
121	/*			
122	vec_handler_end();			
123	Restores all registers (except stack pointer), and returns from exception.			
124	Should be the last statement in exception handlers.			
125	*/			
126	#define vec_handler_end() \			
127	asm("moveml sp@-,90x/rrr"); asm("rts")			
128				
129	typedef struct			
130	{			
131	u_long s_00,			
132	u_long s_01,			
133	u_long s_02,			
134	u_long s_03,			
135	u_long s_04,			
136	u_long s_05,			
137	u_long s_06,			
138	u_long s_07,			
139	u_long s_08,			
140	u_long s_09,			
141	u_long s_10,			
142	u_long s_11,			
143	u_long s_12,			
144	u_long s_13,			
145	u_long s_14,			
146	u_long s_15,			
147	u_long sp; /* not really saved, but here for sake of comparing			
148	u_long sr; /* last breakpoint's sp to current breakpoint's */			
149	} vec_saved_registers_struct;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/device.h	DATE 5/23/89	PAGE # 1/16
LINE #	HEADER TEXT			
1	/* SOCS ID: device.h rev 3.1.1, 4/24/89 at 07:33:09 */			
2	/* Device Description Data Structures */			
3	#include "common.h"			
4	/* Define LOW (0x1) /* Used in TIMING_SPEC */			
5	/* Define HIGH (0x2) */			
6	/* Define VALID (LOW HIGH) */			
7	/* Define FLOAT (VALID FLOAT) */			
8	/* Define ANY (0) /* Used in PIN_SPEC */			
9	/* Define GATED_ON (0) /* Used in PIN_SPEC */			
10	/* Define ALWAYS_ON (1) */			
11	/* Define DRIVE_OFF (0) */			
12	/* Define DRIVE_ON (0) */			
13	/* Define DATA (1) */			
14	/* Define EVAL (2) */			
15	/* Define STORE (0) */			
16	/* Define HOME (0) */			
17	/* Define IN (2) */			
18	/* Define OUT (2) */			
19	/* Define IO (3) */			
20	/* Define POWER (5) */			
21	/* Define GROUND (6) */			
22	/* Define NC (0) */			
23	/* Define NO_DRIVE (0) */			
24	/* Define H_DRIVE (1) */			
25	/* Define M_DRIVE (2) */			
26	/* Define M_DRIVE (3) */			
27	/* Define MRE (0) */			
28	/* Define DMRE (1) */			
29	/* Define RO (2) */			
30	/* Define RI (3) */			
31	/* Define RC (4) */			
32	/* Define PUBLIC (0) /* Used in DEVICE_SPEC */			
33	/* Define PRIVATE (1) */			
34	/* Define FEEDBACK_RISE (0) */			
35	/* Define FEEDBACK_FALL (1) */			
36	/* Define INTERNAL (0) */			
37	/* Define EXT1 (1) */			
38	/* Define EXT2 (2) */			
39	typedef struct			
40	{			
41	/* triple of minimum, typical, and maximum */			
42	/* delays, setups, holds, and pulse widths */			
43	/* as extracted from the device's SMD, ps			
44	/* MIN_TTP_MAX			
45	/* major pin number is implicit in the table index */			
46	/* minor pin number */			
47	/* space: 4 */			
48	/* major_state: 4 */			
49	/* minor_state: 4 */			
50	/* state index */			
51	/* TIMING_SPEC */			
52	/* The SMD pin number is implicit in the table index */			
53	{			
54	/* pin_name */			
55	/* pin_number */			
56	/* pin_alias */			
57	/* delay_table */			
58	/* count of pin delays */			
59	/* trace_delay */			
60	/* trace_delay for this signal: ps			
61	/* a_drive_low: 4 */			
62	/* 16 discrete out-drive low current */			
63	/* direction: 1 */			
64	/* direction pin driver: HOME, DMRE, etc			
65	/* clk_edge: 1 */			
66	/* index into edge table: range 0-3 */			
67	/* pin_class: 2 */			
68	/* index into edge table: range 0-3 */			
69	/* clk_format: 1 */			
70	/* format of clock: DMRE, RO, RI, RC			
71	/* io_drive: 1 */			
72	/* NO_DRIVE, H_DRIVE, M_DRIVE, M_DRIVE			
73	/* last_cyc_drive: 1 */			
74	/* NO_DRIVE, H_DRIVE, M_DRIVE, M_DRIVE			
75	/* diff_edge: 1 */			
76	/* 0 selects edge(0), 1 selects edge(1) */			
77	/* h_drive: 1 */			
78	/* hand-drive enable: DRIVE_OFF, DRIVE_ON			
79	/* h_drive: 1 */			
80	/* set to 1 iff pin is used in feedback */			
81	/* kept_alive: 1 */			
82	/* set to 1 iff pin is hooked to keepalive */			
83	/* pulled_up: 1 */			
84	/* set to 1 iff pin is pulled up */			
85	/* pulled_down: 1 */			
86	/* set to 1 iff pin is pulled down */			
87	/* 16 bytes: 15 bits spare */			
88	/* PIN_SPEC */			
89	{			
90	/* pre_bits */			
91	/* pre-ff init sequence bit pattern */			
92	/* fb_bits */			
93	/* feedback init sequence bit pattern */			
94	/* post_bits */			
95	/* post-ff init sequence bit pattern */			
96	/* seq_str */			
97	/* initialization sequence source string */			
98	/* pin_number */			
99	/* SMD pin number for this init sequence */			
100	/* SEQ_SPEC */			
101	{			
102	/* device_name */			
103	/* name of device */			
104	/* module_name */			
105	/* name of module device is in */			
106	/* extra_data */			
107	/* extra data, not from device's SMD */			
108	/* pin_table */			
109	/* table of device pins */			
110	/* seq_table */			
111	/* table of seq. sequences */			
112	/* timing_table */			
113	/* table of timing specifications */			
114	/* MIN_TTP_MAX */			
115	/* table of unique M/T/N triplets */			
116	/* struct device */			
117	/* linked list of devices */			
118	/* MIN_TTP_MAX */			
119	/* default delay triple */			
120	{			
121	/* clock_period: 1 */			
122	/* longest period of clock: ps			
123	/* two_state_sample: 1 */			
124	/* 2-state pin sample time: ps			
125	/* five_state_sample: 1 */			
126	/* 5-state pin sample time: ps			
127	/* setup_time: 1 */			
128	/* default data setup time: ps			
129	/* hold_time: 1 */			
130	/* default data hold time: ps			
131	/* u_short */			
132	/* edge_show_time: 1 */			
133	/* constant edge-to-edge show: ps			
134	/* sd_settling_time: 1 */			
135	/* medium drive settling time: ps			
136	/* hd_settling_time: 1 */			
137	/* hand drive settling time: ps			
138	/* vth, vth, vth */			
139	/* var and not-drive supplies: of			
140	/* vth, vth, vth */			
141	/* tri-state voltage thresholds: of			
142	/* vth, vth, vth */			
143	/* logic-level voltage thresholds: of			
144	/* pin_cnt */			
145	/* count of device pins */			
146	/* post_seq_len */			
147	/* length of post-feedback sequence */			
148	/* fb_seq_len */			
149	/* length of feedback sequence */			
150	/* seq_cnt */			
151	/* count of pins in reset sequence */			
152	/* timing_cnt */			
153	/* count of entries in timing table */			
154	/* state_cnt */			
155	/* count of entries in M/T/N table */			
156	/* clock_type: 1 */			
157	/* type of clock: INTERNAL, EXT1, EXT2 */			
158	/* device_type: 1 */			
159	/* type of device: PUBLIC or PRIVATE */			
160	/* fb_type: 1 */			
161	/* type of feedback: RISE or FALL */			
162	/* use_default: 1 */			
163	/* 1 iff default delay is specified */			
164	/* report_min: 1 */			
165	/* 1 iff missing delays are to be reported */			
166	/* report_max: 1 */			
167	/* 1 iff io state changes are to be reported */			
168	/* ultra_fast: 1 */			
169	/* ultra fast mode if set */			
170	/* dis_timing_check: 1 */			
171	/* disable timing checking */			
172	/* inh_timing_measure: 1 */			
173	/* inhibit timing measurement */			
174	/* DEVICE_SPEC */			
175	{			
176	/* 104 bytes, 6 bits spare */			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/diag_setjmp.h	DATE 5/23/89	PAGE # 1/17
			TIME 1:20:17 pm	
HEADER TEXT				
LINE #				
1	/* SOCS_ID: diag_setjmp.h rev:3.1, 4/24/89 at 07:33:11 */			
2				
3	typedef int jmp_buf[15]; /* pc, lm_current_depth, onstack; 62-7, 62-7 */			
4	extern long lm_current_depth;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/duart.h	DATE 5/23/89	PAGE # 1/18
LINE #	HEADER TEXT			
1	/* SCCS ID: duart.h rev 3.1, 4/24/89 at 07:33:14 */			
2	/*			
3	/* WRITE REGS FOR THE 8255 DUART */			
4	/*			
5	#define MODE_REG_1_A 0 /* offset of mode reg 1 */			
6	#define MODE_REG_2_A 0 /* offset address of mode reg 2 */			
7	#define CLK_SEL_REG_A 4 /* offset clock sel reg */			
8	#define CMD_REG_A 8 /* offset command register */			
9	#define TX_REG_A 0x0c /* transmit register */			
10	#define AUX_CNTRL_REG 0x10 /* offset of auxiliary control register */			
11	#define INT_MASK_REG 0x14 /* offset of interrupt mask reg */			
12	#define CNT_TMR_UPPER_REG 0x18 /* counter/timer upper */			
13	#define CNT_TMR_LOWER_REG 0x1c /* counter/timer lower */			
14	#define MODE_REG_1_B 0x20 /* offset of mode reg 1 */			
15	#define MODE_REG_2_B 0x20 /* offset address of mode reg 2 */			
16	#define CLK_SEL_REG_B 0x24 /* offset clock sel reg */			
17	#define CMD_REG_B 0x28 /* offset command register */			
18	#define TX_REG_B 0x2c /* transmit register */			
19	#define OUT_PORT_REG 0x34 /* offset of output port reg */			
20	#define SET_O_P_PORT 0x38 /* set output port bits */			
21	#define RES_O_P_PORT 0x3c /* reset output port bits */			
22	/*			
23	/* READ REGS FOR 8255 */			
24	/* note: some of the above regs are r/w. */			
25	/* see Data Sheet, for more details */			
26	/*			
27	#define STAT_REG_A 4 /* status reg */			
28	#define RCV_REG_A 0x0c /* rcvr */			
29	#define IN_PORT_CFG 0x10 /* input port change reg */			
30	#define INT_STAT_REG 0x14 /* interrupt status reg */			
31	#define CNT_TMR_UPPER 0x18 /* counter/timer upper */			
32	#define CNT_TMR_LOWER 0x1c /* counter/timer lower */			
33	#define STAT_REG_B 0x24 /* status reg b */			
34	#define RCV_REG_B 0x2c /* rcvr b */			
35	#define INPUT_PORT 0x34 /* input port */			
36	#define START_CNTR 0x38 /* start counter */			
37	#define STOP_CNTR 0x3c /* stop counter */			
38	/*			
39	/* Global DUART turn off/ turn on */			
40	/*			
41	#define DUART_DISABLE 0 /* disable function */			
42	#define DUART_ENABLE 1 /* enable function */			
43	/*			
44	/* Mode reg 1 equates */			
45	/*			
46	#define RX_INT_READY 0 /* rcvr ready int */			
47	#define RX_INT_FFULL 1 /* rfr full int */			
48	/*			
49	#define ERR_MODE_CHAR 0 /* error mode char */			
50	#define ERR_MODE_BLOCK 1 /* block err mode */			
51	/*			
52	/* this combines parity type and parity mode */			
53	/*			
54	#define PARITY_ODD 1 /* odd parity */			
55	#define PARITY_EVEN 0 /* even parity */			
56	#define PARITY_NONE 4 /* no parity */			
57	#define PARITY_FORCED 2 /* parity = 1 */			
58	/*			
59	#define BITS_PER_CHAR_5 0 /* 5 bits per char */			
60	#define BITS_PER_CHAR_6 1 /* 6 bits per char */			
61	#define BITS_PER_CHAR_7 2 /* 7 bits per char */			
62	#define BITS_PER_CHAR_8 3 /* 8 bits per char */			
63	/*			
64	/* Mode reg 2 */			
65	/*			
66	#define CH_MODE_NORMAL 0 /* normal channel */			
67	#define CH_MODE_AUTO_ECHO 1 /* auto echo char */			
68	#define CH_MODE_LOCAL_LOOP 2 /* local loop */			
69	#define CH_MODE_REMOTE_LOOP 3 /* remote loop */			
70	/*			
71	#define STOP_BITS_HALF 0 /* half a stop bit */			
72	#define STOP_BITS_1 7 /* 1 stop bit */			
73	#define STOP_BITS_2 0xf /* 2 stop bits */			
74	/*			
75	/* Clock Select */			
76	/*			
77	#define BAUD_110 1 /* Baud rate 110 */			
78	#define BAUD_300 4 /* Baud rate 300 */			
79	#define BAUD_1200 6 /* Baud rate 1200 */			
80	#define BAUD_2400 8 /* Baud rate 2400 */			
81	#define BAUD_4800 9 /* Baud rate 4800 */			
82	#define BAUD_9600 0xb /* Baud rate 9600 */			
83	#define BAUD_96_12_24 (BAUD_9600 0x10) /* Baud rate 9600/1200/2400 */			
84	#define BAUD_24_96_12 (BAUD_2400 0x10) /* Baud rate 2400/9600/1200 */			
85	/*			
86	/* command reg */			
87	/*			
88	#define NOP 0 /* no op */			
89	#define RESET_MODE_REG 1 /* reset mode reg ptr */			
90	#define RESET_RCVR 2 /* reset rcvr */			
91	#define RESET_TX 3 /* reset tx */			
92	#define RESET_ERR 4 /* reset error */			
93	#define RESET_CN 5 /* reset channel */			
94	/*			
95	/* Mode reg 1 */			
96	/*			
97	typedef struct			
98	{			
99	unsigned			
100	rx_ctl:1, /* RX RTS CONTROL */			
101	rx_int:1, /* RX INT SELECT */			
102	error_mode:1, /* error mode */			
103	parity:1, /* parity mode & parity type */			
104	bits_per_char:2, /* bits per char */			
105	pad:24, /* 24 bit pad */			
106	} MODE_REG_1;			
107	/*			
108	/* Mode reg 2 */			
109	/*			
110	typedef struct			
111	{			
112	}			
113	/*			
114	/*			
115	/*			
116	/*			
117	/*			
118	/*			
119	/*			
120	/*			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/duart.h	DATE 5/23/89 TIME 1:20:17 pm	PAGE # 2/19
LINE #	HEADER TEXT			
121	/* channel mode */			
122	/* tx rts */			
123	/* enable cts */			
124	/* stop bits */			
125	/* 24 bit pad */			
126	/*			
127	MODE_REG,			
128	/*			
129	/* clock select			
130	/*			
131	typedef struct			
132	{			
133	/*			
134	/* rx clk sel */			
135	/* tx clk sel */			
136	/* 24 bit pad */			
137	/*			
138	CLK_SEL,			
139	/*			
140	/* command reg:			
141	/*			
142	typedef struct			
143	{			
144	/*			
145	/* force zero */			
146	/* command */			
147	/* disable tx */			
148	/* enable tx */			
149	/* disable rx */			
150	/* enable rx */			
151	/* 24 bit pad */			
152	/*			
153	CMD_REG,			
154	/*			
155	/* STATUS reg			
156	/*			
157	typedef struct			
158	{			
159	/*			
160	/* rxd break */			
161	/* framing error */			
162	/* parity error */			
163	/* receiver error */			
164	/* tx empty */			
165	/* tx ready */			
166	/* cifo full */			
167	/*			
168	/* rx ready */			
169	/*			
170	STATUS_REG,			
171	/*			
172	/*			
173	/* The ACR control reg. is one we write zeros in.			
174	/*			
175	typedef struct			
176	{			
177	/*			
178	/*			
179	/*			
180	/*			
181	/*			
182	/*			
183	/*			
184	/*			
185	/* input port change */			
186	/* delta break b */			
187	/* RRDY / FULL */			
188	/* TX RDY */			
189	/* counter */			
190	/* delta break a */			
191	/* RRDY / FULL */			
192	/* TX RDY */			
193	/*			
194	/*			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/eprom.h	DATE 5/23/89	PAGE # 1/20
LINE #		HEADER TEXT		
1		/* SCCS ID: eprom.h rev 2.1, 4/24/89 at 07:33.15 */		
2		/* NOTES: */		
3		/* We don't need an explicit DAB size as that information is easily		
4		/* calculated from "configuration" by counting the number of DAB slots.		
5		/* All DAB information is an even number of bytes long. This facilitates		
6		/* the reading and writing of the DAB EPROM as it is 64 bytes by 16 bits		
7		/* and not 128 bytes by 8 bits.		
8		/*		
9		#define DAB_EPROM_VERSION 0x2B /* used with "version number" below, start		
10		/* somewhere other than 0 or 1. */		
11		#define NAME_LENGTH 20 /* length of "device name" */		
12		#define MAKER_LENGTH 16 /* length of "maker" */		
13		#define MAN_LENGTH 16 /* length of "manufacturer" */		
14		#define REV_LENGTH 4 /* length of "revision" */		
15		#define TYPE_LENGTH 8 /* length of "type" */		
16		/*		
17		/* THIS STRUCTURE MUST ALWAYS BE 128 BYTES.		
18		/* THE WRITE COUNT AND CHECKSUM ARE THE LAST 2		
19		/* FIELDS. THIS IS BECAUSE IF THE STRUCTURE CHANGES		
20		/* THE WRITE COUNT IS STILL REMAINABLE.		
21		/* THE CHECKSUM IS THE LAST FIELD, FOR CONSISTENCY.		
22		/* THE CHECKSUM & WRITE COUNT OCCUPY THE LAST 2		
23		/* FIELDS. IF ANY ADDITIONS OR DELETIONS OCCUR		
24		/* CHANGE THE RESERVED FIELD APPROPRIATELY.		
25		/*		
26		typedef struct {		
27		char signature[4]; /* INHX */		
28		u_short version; /* version of this information structure */		
29		char device_name[NAME_LENGTH]; /* name of device */		
30		char model_maker[MAKER_LENGTH]; /* name of the maker of this model */		
31		char model_revision[REV_LENGTH]; /* examples: 1.0, 2.0, 3.0 */		
32		char dab_manufacturer[MAN_LENGTH]; /* name of the maker of this DAB */		
33		char dab_revision[REV_LENGTH]; /* examples: 1.0, 2.0, 3.0 */		
34		char dab_type[TYPE_LENGTH]; /* examples: DIP, PGA, LCC */		
35		u_short location_count; /* number of times the DAB has been inserted		
36		u_short segment_number; /* 16 bit code indicating a DAB "segment" */		
37		/* number for multiple physical DABs making		
38		/* up a single device. */		
39		u_long configuration; /* 32 bit code indicating the physical DAB		
40		/* configuration based upon 4 lines X 8 slots. */		
41		u_char reserved[42]; /* for future use. */		
42		u_short write_count; /* number of times the EPROM has been written */		
43		u_short checksum; /* checksum over all the other DAB information */		
44		} DAB_EPROM;		
45		#define SIG_WORD_0 (u_short)('M' + ('L' << 8))		
46		#define SIG_WORD_1 (u_short)('I' + ('S' << 8))		
47		#define EPROM_SIGNATURE_0 0		
48		#define EPROM_SIGNATURE_1 1		
49		#define EPROM_WRITE_COUNT 62		
50		#define EPROM_CHECKSUM 63		
51		#define CPU_EPROM_SIZE 128		
52		/*		
53		/* Equates for the offsets		
54		/*		
55		#define E2_SIGNATURE 0		
56		#define E2_VER 4		
57		#define E2_DEV_NAME (E2_VER + 2)		
58		#define E2_MODEL_MAKER (E2_DEV_NAME + NAME_LENGTH)		
59		#define E2_MODEL_REV (E2_MODEL_MAKER + MAKER_LENGTH)		
60		#define E2_DAB_MAN (E2_MODEL_REV + REV_LENGTH)		
61		#define E2_DAB_REV (E2_DAB_MAN + MAN_LENGTH)		
62		#define E2_DAB_TYPE (E2_DAB_REV + REV_LENGTH)		
63		#define E2_INS_COUNT (E2_DAB_TYPE + TYPE_LENGTH)		
64		#define E2_SEG_NUMBER (E2_INS_COUNT + 2)		
65		#define E2_CONFIG (E2_SEG_NUMBER + 2)		
66		/*		
67		/* This are the options		
68		/*		
69		#define START_OPTION 0x1		
70		#define EPROM_READ 0x2		
71		#define EPROM_WRITE 0x1		
72		#define EPROM_ERASE 0x1		
73		#define EPROM_ALL 0x0		
74		#define EPROM_REG_OPTION 0x00 /* 16 and option use register */		
75		#define EPROM_EVEN 0x10		
76		#define EPROM_ODDS 0x00		
77		#define EPROM_REAL 0x10		
78		#define EPROM_VIRTUAL 0x10		
79		#define SET_CLK_0 0x100		
80		#define SET_CLK_1 0x101		
81		#define TOG_CLK 0x102		
82		#define NO_TOG_CLK 0x103		
83		#define SET_CS_0 0x104		
84		#define SET_CS_1 0x105		
85		#define TEST_OPCODE 0x107		
86		#define READ_DATA 0x108		
87		#define WRITE_DATA 0x109		
88		#define EPROM_START 0x10a		
89		#define WAIT_OUT_LOW 0x10b		
90		#define WAIT_OUT_HIGH 0x10c		
91		#define WAIT_IN_LOW 0x10d		
92		#define WAIT_IN_HIGH 0x10e		
93		#define NOP 0x10f		
94		#define DONE 0x110		
95		#define SET_DATA_IN_LOW 0x111		
96		#define EPROM_DELAY 0x112		
97		#define LITERAL 0xc000		

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/eprom.h	5/23/89	1/21
			TIME 1:20:18 pm	
LINE #	HEADER TEXT			
1	/* SCCS ID: eprom.h rev 3.1.1, 4/24/89 at 07:33:21 */			
2	/*			
3	* This structure defines the organization of EPROM. It begins through			
4	* the beginning of actual code. Note that this structure should reflect the			
5	* organization of keeem.h and boot.c.			
6	*/			
7				
8	typedef struct			
9	{			
10	u_long reset_ep; /* reset vector, initial EP */			
11	u_long reset_pc; /* reset vector, initial PC */			
12	u_long checksum; /* EPROM checksum */			
13	u_long diagboot_offset; /* offset of Diagboot object */			
14	u_long diagboot_entry; /* entry point for Diagboot */			
15	u_long eprom_size; /* total number of bytes actually used */			
16	u_long eprom_size; /* physical size of EPROM in bytes */			
17	char revision[64]; /* revision/identifier string */			
18	} eprom_struct;			
19				
20	/*			
21	* Various constants referring to the previous structure:			
22	*/			
23	#define EPROM_WOChecksum_START 0x01			
24	#define EPROM_WOChecksum_END 0x0C			
25	#define EPROM_CODE_OFFSET (sizeof(eprom_struct))			
26	#define EPROM_REVISION_LENGTH 64			
27				
28	/*			
29	* Seeds checksum calculation (pretty random)			
30	*/			
31	#define EPROM_CHECKSUM_INIT 0x7be9c1bd			
32				
33	/*			
34	* Default eprom_size (for use by keeprom) --- this assumes 2712's			
35	*/			
36	#define EPROM_SIZE_DEFAULT (2 * 512 * 1024 / 8)			
37				
38	/*			
39	* Where to copy the top part of EPROM to in DRAM			
40	*/			
41	#define DIAGBOOT_START 0x300000			
42				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/fifo.h	DATE 5/23/89 TIME 1:20:18 pm	PAGE # 1/22
LINE #	HEADER TEXT			
1	/* SOCS_ID: fifo.h rev 3.1, 4/24/89 at 07:33:24 */			
2	/*			
3	/* fifo.h			
4	/* fifo header for CPU board.			
5	/*			
6	/*			
7	/* max fifos in the system.			
8	/*			
9	/* max fifo entries in the system.			
10	/*			
11	#define MAX_FIFOS 4			
12	/*			
13	/* max fifo entries in the system.			
14	/*			
15	#define MAX_FIFO_ENTRIES 300			
16	#define MIN_TASK_ENTRIES 100			
17	/*			
18	/* fifo header type			
19	/*			
20	struct fifo_header_type			
21	{			
22	unsigned char user;			
23	unsigned char task;			
24	struct fifo_header_type *next;			
25	char *data;			
26	};			
27	/*			
28	/* per fifo information			
29	/*			
30	struct fifo_type			
31	{			
32	int count;			
33	struct fifo_header_type *head;			
34	struct fifo_header_type *tail;			
35	};			
36	/*			
37	/*			
38	struct fifo_entry			
39	{			
40	char *data;			
41	unsigned char task;			
42	unsigned char user;			
43	char fifo_no;			
44	};			
45	/*			
46	/*			
47	/* The fifos			
48	/*			
49	#define HEADER_FIFO 0			
50	#define RX_FIFO 1			
51	#define TX_FIFO 2			
52	/*			
53	/*			
54	/* TASK defines:			
55	/*			
56	#define NO_TASK 0			
57	#define ISR_TASK 0x01			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/hardware.h	DATE 5/23/89	PAGE # 1/23
LINE #		HEADER TEXT		
1		/* SOCS ID: hardware.h rev 3.1, 4/24/89 at 07:33:27 */		
2		/* This header file contains all of the system physical addresses as specified		
3		in the document "M-1000 Processor Maps" dated December 8, 1987.		
4		*/		
5				
6				
7		#define CPU_TIMER_ADDR 0x10c30000		
8		#define CPU_ID_WREN_ADDR 0x10c50000		
9				
10		#define CPU_STATUS_REG_ADDR 0x10c60000		
11		#define CPU_DMC_WRITE_MASK 0xffffffff		
12		#define CPU_DFP_SWITCH1_MASK 0x7fffffff		
13		#define CPU_DFP_SWITCH2_MASK 0xbfffffff		
14		#define CPU_DFP_SWITCH3_MASK 0xdfffffff		
15		#define CPU_DFP_SWITCH4_MASK 0xfefeffff		
16		#define SOFTWARE_RESET_MASK 0xffffffff		
17				
18		#define LAME_0_START_ADDR 0x00000000		
19		#define LAME_ADDR_INC 0x10000000		
20				
21		#define CLOS_START_ADDR 0x00000000		
22				
23		#define LAME_ADDR_MASK 0x00000000		
24		#define LAME_SEGMENT_A_OFFSET 0x0		
25		#define LAME_SEGMENT_B_OFFSET 0x4000000		
26		#define LAME_SEGMENT_C_OFFSET 0x8000000		
27		#define LAME_LINK_TABLE_OFFSET 0xc000000		
28		#define LAME_PAC_START_OFFSET 0x1000000		
29		#define LAME_PEL_0_START_OFFSET 0x0000000		
30		#define LAME_PEL_ADDR_INC 0x1000		
31				
32		#define PAC_BOARD_ID_OFFSET 0x0		
33		#define PAC_PAN_COUNT_REG_OFFSET 0x80		
34		#define PAC_PAN_CONFIG_REG_OFFSET 0x100		
35		#define PAC_BRANCH_ADDR_REG_OFFSET 0x180		
36		#define PAC_CLOCK_SPEED_REG_OFFSET 0x200		
37		#define PAC_ERROR_REG_OFFSET 0x280		
38		#define PAC_BLOCK_OFFSET 0x300		
39		#define PAC_PARITY_ERROR_ADDR_OFFSET 0x380		
40		#define PAC_PAN_0_PRESENT_REG_OFFSET 0x400		
41		#define PAC_PAN_0_ID_SIZE_REG_OFFSET 0x480		
42		#define PAC_PAN_1_PRESENT_REG_OFFSET 0x500		
43		#define PAC_PAN_1_ID_SIZE_REG_OFFSET 0x580		
44		#define PAC_PAN_2_PRESENT_REG_OFFSET 0x600		
45		#define PAC_PAN_2_ID_SIZE_REG_OFFSET 0x680		
46		#define PAC_PAN_3_PRESENT_REG_OFFSET 0x700		
47		#define PAC_PAN_3_ID_SIZE_REG_OFFSET 0x780		
48		#define PAC_CONFIG_HIGH_WORD_PARITY_MASK 0xffffffff		
49		#define PAC_CONFIG_LOW_WORD_PARITY_MASK 0xffffffff		
50				
51		#define PAC_PAN_PRESENT_REG_ADDR_INC 0x100		
52		#define PAC_PAN_ID_SIZE_REG_ADDR_INC 0x100		
53		#define PAC_LOW_WORD_PARITY_MASK 0x7fffffff		
54		#define PAC_HIGH_WORD_PARITY_MASK 0xbfffffff		
55		#define PAC_PARITY_ADDR_MASK 0xc0000000		
56				
57		/* 2 HEX positions indicates the size of the PAN */		
58		#define PAC_PAN_BOARD_ID_MASK 0xffffffff		
59		#define PAC_PAN_SIZE_SHIFT 30		
60		#define PAC_PAN_128K 0		
61		#define PAC_PAN_512K 1		
62		#define PAC_PAN_2M 2		
63		#define BLOCKS_IN_128K 512		
64		#define BLOCKS_IN_512K 2048		
65		#define BLOCKS_IN_2M 8192		
66				
67		#define PAC_PAN_COUNT_EQ_0 0x0		
68		#define PAC_PAN_COUNT_EQ_1 0x1		
69		#define PAC_PAN_COUNT_EQ_2 0x3		
70		#define PAC_PAN_COUNT_EQ_3 0x7		
71		#define PAC_PAN_COUNT_EQ_4 0xf		
72				
73		#define PEL_NC_0_OFFSET 0x0		
74		#define PEL_NC_ADDR_INC 0x100		
75				
76		#define PEL_VIL_DAC_WRITE_OFFSET 0x500		
77		#define PEL_VIE_DAC_WRITE_OFFSET 0x504		
78		#define PEL_VILH_DAC_WRITE_OFFSET 0x600		
79		#define PEL_VILH_DAC_WRITE_OFFSET 0x604		
80		#define PEL_VIEH_DAC_WRITE_OFFSET 0x700		
81		#define PEL_VIEH_DAC_WRITE_OFFSET 0x704		
82		#define PEL_VIL_DAC_READ_OFFSET 0x800		
83		#define PEL_VIE_DAC_READ_OFFSET 0x804		
84		#define PEL_VILH_DAC_READ_OFFSET 0x900		
85		#define PEL_VIEH_DAC_READ_OFFSET 0x904		
86		#define PEL_VIEH_DAC_READ_OFFSET 0x910		
87		#define PEL_VIEH_DAC_READ_OFFSET 0x914		
88		#define PEL_OTVCC_DAC_READ_OFFSET 0x918		
89		#define PEL_DAMNICAL_DAC_READ_OFFSET 0x91c		
90		#define PEL_ID_PROM_OFFSET 0x900		
91		#define PEL_STATUS_CONTROL_OFFSET 0x900		
92				
93		#define MAX_PEL_DRIVE_CURRENT 1000 /* 1000 mA */		
94				
95		#define PEL_DRIVE_HIGH_3_CURRENT 20 /* 2.0 mA */		
96		#define PEL_DRIVE_HIGH_2_CURRENT 12 /* 1.2 mA */		
97		#define PEL_DRIVE_HIGH_1_CURRENT 5 /* 0.5 mA */		
98		#define PEL_DRIVE_HIGH_0_CURRENT 1 /* 0.1 mA */		
99				
100		#define PEL_DRIVE_LOW_3_CURRENT 30 /* 3.0 mA */		
101		#define PEL_DRIVE_LOW_2_CURRENT 12 /* 1.2 mA */		
102		#define PEL_DRIVE_LOW_1_CURRENT 5 /* 0.5 mA */		
103		#define PEL_DRIVE_LOW_0_CURRENT 2 /* 0.2 mA */		
104				
105		#define DAC_TIMER_VALUE 900 /* microseconds */		
106		#define TRG_TIMER_VALUE 500 /* microseconds */		
107				
108		#define NC_REG_0_OFFSET 0x0		
109		#define NC_REG_ADDR_INC 0x4		
110				
111		#define NC_DATA_OUT_REG_OFFSET 0x0		
112		#define NC_MODE_REG_OFFSET 0x4		
113		#define NC_EDOC_REG_OFFSET 0x8		
114		#define NC_CTL_OUT_REG_OFFSET 0xc		
115		#define NC_LAST_CYCLE_REG_OFFSET 0x10		
116		#define NC_WSDEN_REG_OFFSET 0x14		
117		#define NC_DATA_FORMAT_REG_OFFSET 0x18		
118		#define NC_ERROR_DATA_REG_OFFSET 0x1c		
119		#define NC_SHORT_SAMPLE_REG_OFFSET 0x20		
120		#define NC_U_SAMPLE_REG_OFFSET 0x24		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/hardware.h	DATE 5/23/89	PAGE # 2/24
LINE #		HEADER TEXT		
21		#define MC_EII_SAMPLE_REG_OFFSET 0x28		
22		#define MC_VALUE_SAMPLE_REG_OFFSET 0x2c		
23		#define MC_TINTING_SAMPLE_REG_OFFSET 0x30		
24		#define MC_UNUSED_REG_OFFSET 0x34		
25		#define MC_STATUS_OFFSET 0x38		
26		#define MC_RESET_REG_OFFSET 0x3c		
27				
28		/* The following addresses are not defined yet: */		
29		#define CPU_MEM_SIZE_ADDR 0x0100000		
30		#define CPU_SLOT_COUNT_ADDR 0x1000004		
31		#define CPU_LANE_COUNT_ADDR 0x1000008		
32		#ifndef DBASE		
33		/* These are dummy locations for these entities for data base purposes. */		
34		#define PEL_DAB_EEPROM_OFFSET 0xb00		
35		#endif		
36		#define CLOS_BOARD_ID_OFFSET 0x0		
37				
38		#define MAX_EDGE_COUNT 7		
39				
40		#define MAX_SEGMENT_PER_DEVICE 32		
41				
42		/* Feedback pipeline delay */		
43		#define FB_PIPE_DELAY 8		
44				
45		/* Constants related to edge placement */		
46		#define MAX_MODELER_CLOCK_PERIOD 666666 /* 150KHz in picoseconds */		
47		#define MIN_MODELER_CLOCK_PERIOD 40000 /* 25 MHz in picoseconds */		
48		#define MAGIC_OUTPUT_DEAD_TIME 5000 /* picosec */		
49		#define EDOUT_OFFSET 15000 /* picosec */		
50		#define EDOUT_TO_STORE_OFFSET 10000 /* picosec */		
51				
52		/* For I/O where pipe running in ultra fast mode */		
53		#define EDOUT_PERIOD_THRESHOLD 1000000 /* ps */		
54		#define EDOUT_HOLD_TIME_THRESHOLD 200000 /* ps */		
55		#define STORE_TO_EDOUT_OFFSET 20000 /* ps */		
56				
57		#define NO_TM_RISING_SAMPLE_TIME 125000 /* picosec */		
58		#define MAX_SAMPLE_TIME_RANGE 1000000 /* picosec */		
59		#define MIN_SAMPLE_PULSE_WIDTH 600000 /* picosec hardware constraint */		
60		/* The following constant has to be greater than MAX_SAMPLE_TIME_RANGE */		
61		#define MIN_SAMPLE_PULSE_WIDTH		
62				
63		#define DEFAULT_FALL_SAMPLE_TIME 1000000 /* picosec */		
64		#define DEFAULT_FALL_SAMPLE_TIME2 5000000 /* picosec */		
65				
66		#define MAX_PAN_COUNT 4		
67				
68		#define MAX_DOT_VCC_VOLTAGE 6000 /* mV */		
69		#define MAX_ADC_VOLTAGE 5000 /* mV */		
70				
71		#define EXT0_REG_VALUE 3		
72		#define EXT1_REG_VALUE 4		
73				
74		/* These 3 constants are also defined in the def.h */		
75		#define EARLYSAMPLETRIGGERMODE 0		
76		#define EDGE7SAMPLETRIGGERMODE 1		
77				
78		#ifndef DBASE		
79		#define MAX_NUM_ID_CHAR 16		
80		#define MAX_DAB_TYPE_CHAR 4		
81		#define MAX_REVISION_CHAR 4		
82		#define MAX_PART_NAME_CHAR 20		
83				
84		typedef struct {		
85		u_short stat's_info; /* for database management */		
86		u_short value_type; /* for database management */		
87		char max_id[MAX_NUM_ID_CHAR];		
88		char dab_type[MAX_DAB_TYPE_CHAR];		
89		char revision[MAX_REVISION_CHAR];		
90		char part_name[MAX_PART_NAME_CHAR];		
91		u_long dab_shape;		
92		u_short dab_segment_number;		
93		EEPROM_CONTENTS;		
94		#endif		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/id.h	DATE 5/23/89	PAGE # 1/25
LINE #		HEADER TEXT		
1		/* SCCS ID: id.h rev 3.1, 4/24/89 at 07:33:38 */		
2		/*		
3		* Define various LMS hardware ID PROMs		
4		*/		
5		/*		
6		* Define ID PROM bytes		
7		* Define ID CHECKSUM		
8		* Define ID CHECKSUM INIT		
9		* Define ID CHECKSUM GOOD		
10		* Define ID CHECKSUM OFFSET		
11		* Define ID NULL DATA		
12		* Define ID NULL DATA		
13		* Define ID NULL DATA		
14		* Define ID NULL DATA		
15		* Define ID NULL DATA		
16		* Define ID NULL DATA		
17		* Define ID NULL DATA		
18		* Define ID NULL DATA		
19		* Define ID NULL DATA		
20		* Define ID NULL DATA		
21		* Define ID NULL DATA		
22		* Define ID NULL DATA		
23		* Define ID NULL DATA		
24		* Define ID NULL DATA		
25		* Define ID NULL DATA		
26		* Define ID NULL DATA		
27		* Define ID NULL DATA		
28		* Define ID NULL DATA		
29		* Define ID NULL DATA		
30		* Define ID NULL DATA		
31		* Define ID NULL DATA		
32		* Define ID NULL DATA		
33		* Define ID NULL DATA		
34		* Define ID NULL DATA		
35		* Define ID NULL DATA		
36		* Define ID NULL DATA		
37		* Define ID NULL DATA		
38		* Define ID NULL DATA		
39		* Define ID NULL DATA		
40		* Define ID NULL DATA		
41		* Define ID NULL DATA		
42		* Define ID NULL DATA		
43		* Define ID NULL DATA		
44		* Define ID NULL DATA		
45		* Define ID NULL DATA		
46		* Define ID NULL DATA		
47		* Define ID NULL DATA		
48		* Define ID NULL DATA		
49		* Define ID NULL DATA		
50		* Define ID NULL DATA		
51		* Define ID NULL DATA		
52		* Define ID NULL DATA		
53		* Define ID NULL DATA		
54		* Define ID NULL DATA		
55		* Define ID NULL DATA		
56		* Define ID NULL DATA		
57		* Define ID NULL DATA		
58		* Define ID NULL DATA		
59		* Define ID NULL DATA		
60		* Define ID NULL DATA		
61		* Define ID NULL DATA		
62		* Define ID NULL DATA		
63		* Define ID NULL DATA		
64		* Define ID NULL DATA		
65		* Define ID NULL DATA		
66		* Define ID NULL DATA		
67		* Define ID NULL DATA		
68		* Define ID NULL DATA		
69		* Define ID NULL DATA		
70		* Define ID NULL DATA		
71		* Define ID NULL DATA		
72		* Define ID NULL DATA		
73		* Define ID NULL DATA		
74		* Define ID NULL DATA		
75		* Define ID NULL DATA		
76		* Define ID NULL DATA		
77		* Define ID NULL DATA		
78		* Define ID NULL DATA		
79		* Define ID NULL DATA		
80		* Define ID NULL DATA		
81		* Define ID NULL DATA		
82		* Define ID NULL DATA		
83		* Define ID NULL DATA		
84		* Define ID NULL DATA		
85		* Define ID NULL DATA		
86		* Define ID NULL DATA		
87		* Define ID NULL DATA		
88		* Define ID NULL DATA		
89		* Define ID NULL DATA		
90		* Define ID NULL DATA		
91		* Define ID NULL DATA		
92		* Define ID NULL DATA		
93		* Define ID NULL DATA		
94		* Define ID NULL DATA		
95		* Define ID NULL DATA		
96		* Define ID NULL DATA		
97		* Define ID NULL DATA		
98		* Define ID NULL DATA		
99		* Define ID NULL DATA		
100		* Define ID NULL DATA		
101		* Define ID NULL DATA		
102		* Define ID NULL DATA		
103		* Define ID NULL DATA		
104		* Define ID NULL DATA		
105		* Define ID NULL DATA		
106		* Define ID NULL DATA		
107		* Define ID NULL DATA		
108		* Define ID NULL DATA		
109		* Define ID NULL DATA		
110		* Define ID NULL DATA		
111		* Define ID NULL DATA		
112		* Define ID NULL DATA		
113		* Define ID NULL DATA		
114		* Define ID NULL DATA		
115		* Define ID NULL DATA		
116		* Define ID NULL DATA		
117		* Define ID NULL DATA		
118		* Define ID NULL DATA		
119		* Define ID NULL DATA		
120		* Define ID NULL DATA		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/id.h	DATE 5/23/89	PAGE # 2/26
			TIME 1:20:18 pm	
LINE #	HEADER TEXT			
121	u_char width,			
122	u_short patterns, /* ((times ID_GEN_PATTERNS)* */			
123	u_char unused[23],			
124	u_char checksum,			
125	} ID_FROM_PAC,			
126	typedef struct			
127	{			
128	id_stuff generic,			
129	u_char speed,			
130	u_char width,			
131	u_short patterns,			
132	u_char unused[23],			
133	u_char checksum,			
134	} ID_FROM_PAN,			
135	typedef struct			
136	{			
137	id_stuff generic,			
138	u_char speed,			
139	u_char width,			
140	u_short patterns, /* i.e. built-in patterns memory */			
141	data_code magic_build, /* Magic chips' lot date code */			
142	u_char unused[21],			
143	u_char checksum,			
144	} ID_FROM_PEL,			
145	/*			
146	Call the following macro with argument equal to the board type to			
147	determine if data should be stored inverted in the FROM when generating			
148	the actual FROM contents.			
149	*/			
150	#define ID_INVERTED(x) ((x == ID_BT_PAN128K) (x == ID_BT_PAN512K) \			
151	(x == ID_BT_PAN2M))			
152				

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/intr.h	5/23/89	1/27
			TIME	1:20:18 pm
LINE #	HEADER TEXT			
1	/* SCCS ID: intr.h rev 3.1. 4/24/89 at 07:33:33 */			
2	#ifndef DIAGS			
3	#define INTR_INIT jmp_buf *save_buf, local_buf, /* local variables */			
4				
5	#define INTR_BEGIN save_buf = diag_jmpbuf, \			
6	if (!diag_setjmp(local_buf)) { \			
7	diag_jmpbuf = (jmp_buf *)local_buf,			
8	} else { diag_jmpbuf = save_buf,			
9	#define INTR_END diag_jmpbuf = save_buf,			
10				
11	extern jmp_buf *diag_jmpbuf,			
12	#else DIAGS			
13	#define INTR_INIT			
14	#define INTR_BEGIN if (1) {			
15	#define INTR_END } else {			
16	#define INTR_GOT_ONE			
17	#define INTR_END }			
18	#endif DIAGS			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lance.h	DATE 5/23/89	PAGE # 1/28
HEADER TEXT				
LINE #	/* SCC3_ID: lance.h rev 3.1, 4/24/89 at 07:33:34 */			
1	/*			
2	The following structure and define definitions are for the			
3	Advanced Micro Devices:			
4	/*			
5	Am7990 Local Area Controller for Ethernet (LANCE);			
6	/*			
7	as taken from the 10/85 LANCE data sheet (publication 905698B/0)			
8	/*			
9	*/			
10	/*			
11	Used to select one of the LANCE's control and status registers.			
12	Writes into the register address port before accessing the			
13	desired register data port.			
14	/*			
15	#define SELECT_CSR0 0			
16	#define SELECT_CSR1 1			
17	#define SELECT_CSR2 2			
18	#define SELECT_CSR3 3			
19	/* CSR1, CSR2, CSR3 are only accessible */			
20	/* when the STOP bit of CSR0 is set. */			
21	/*			
22	*/			
23	/*			
24	Control and Status Register 0			
25	/*			
26	In the comment section:			
27	INT - Interrupt, SE - System Error, R/T-off - Receive/Transmit-off			
28	/*			
29	#define ERROR_SUMMARY 0x0000 /* Read-only ==> Logical OR of all ERRORs */			
30	#define TRANSMIT_BABBLE_ERROR 0x0000 /* Read/Write 1 to clear ==> INT, SE */			
31	#define COLLISION_ERROR 0x0000 /* Read/Write 1 to clear ==> SE (heartbeat) */			
32	#define RISEN_PACKET_ERROR 0x0000 /* Read/Write 1 to clear ==> INT, SE */			
33	#define MEMORY_ERROR 0x0000 /* Read/Write 1 to clear ==> INT, SE, R/T-off */			
34	#define RECEIVE_INTERRUPT 0x0400 /* Read/Write 1 to clear ==> INT */			
35	#define TRANSMIT_INTERRUPT 0x0200 /* Read/Write 1 to clear ==> INT */			
36	#define INITIALIZE_DONE 0x0100 /* Read/Write 1 to clear ==> INT */			
37	#define INTERRUPT_SUMMARY 0x0000 /* Read-only */			
38	#define INTERRUPT_ENABLE 0x0040 /* Complicated */			
39	#define RECEIVE_ON 0x0020 /* Read-only ==> check after every receive */			
40	#define TRANSMIT_ON 0x0010 /* Read-only ==> check after every transmit */			
41	#define TRANSMIT_DEMAND 0x0008 /* No-read/Write 1 to set */			
42	#define STOP_ACTIVITY 0x0004 /* Read/Write 1 to set ==> before START_ACTIVITY */			
43	#define START_ACTIVITY 0x0002 /* Read/Write 1 to set ==> before INIT START */			
44	#define INITIALIZE_START 0x0001 /* Read/Write 1 to set ==> after START_ACTIVITY */			
45	/*			
46	*/			
47	/*			
48	Control and Status Register 1 & 2			
49	/*			
50	These are the low-order 16 bits of the initialization block			
51	address (word aligned) plus 8 bits reserved plus the 8			
52	high order bits of the initialization block address.			
53	/*			
54	See the above comment about accessibility.			
55	/*			
56	*/			
57	/*			
58	Control and Status Register 3			
59	/*			
60	See the above comment about accessibility			
61	/*			
62	#define BYTE_SNAP 0x04 /* BSWP */			
63	#define ALE_CONTROL 0x02 /* ACOW */			
64	#define BYTE_CONTROL 0x01 /* BCOM */			
65	/*			
66	Definitions of who owns the receive or transmit buffer:			
67	Used with "own_buffer" in "message_descriptor."			
68	/*			
69	#define HOST_BUFFER_OWNERSHIP 0			
70	#define LANCE_BUFFER_OWNERSHIP 1			
71	/*			
72	*/			
73	/*			
74	Initialization block structure definition			
75	/*			
76	This structure must be word aligned in system memory.			
77	/*			
78	*/			
79	typedef struct initialization_block {			
80	/* Mode Register (MCR) */			
81	unsigned promiscuous:1, /* FROM */			
82	unsigned reserved_1:8, /* RES */			
83	unsigned internal_loopback:1, /* INTL */			
84	unsigned disable_retry:1, /* DRTY */			
85	unsigned force_collision:1, /* COLL */			
86	unsigned disable_crc_transmit:1, /* DTCK */			
87	unsigned loopback:1, /* LOOP */			
88	unsigned disable_transmit:1, /* DTX */			
89	unsigned disable_receive:1, /* RXE */			
90	/*			
91	*/			
92	/*			
93	Physical Address (PADDR)			
94	/*			
95	48 bit physical Ethernet address accessed via bytes to			
96	handle VM86 access to the FROM that contains this data:			
97	/*			
98	The "strange" byte ordering is extremely important and must not			
99	be changed in order to match up with the LANCE's expectations.			
100	/*			
101	If PADDR[15:0] = 5CBA and PADDR[31:16] = 9876 and PADDR[47:32] = 5432			
102	then the transmitted bit stream is (from first bit to last):			
103	/*			
104	010111010011101011101110100110011001100110011001100110011010			
105	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z			
106	/*			
107	and the printed ASCII Ethernet Address is: BA-50-76-98-32-54.			
108	/*			
109	This means that physical addresses have an even number as the first			
110	printed ASCII Ethernet Address byte and that multicast addresses have			
111	an odd number as the first printed ASCII Ethernet Address byte.			
112	/*			
113	u_char physical_address_2, /* PADDR */			
114	u_char physical_address_1, /* PADDR ==> bit[0] must be 0 */			
115	u_char physical_address_4, /* PADDR */			
116	u_char physical_address_3, /* PADDR */			
117	u_char physical_address_6, /* PADDR */			
118	u_char physical_address_5, /* PADDR */			
119	/*			
120	*/			

Copyright 1989 Logic Modeling Systems	HEADER FILE include/lance.h	DATE 5/23/89 TIME 1:20:18 pm	PAGE # 2/29
	HEADER TEXT		

```
121  * Logical Address Filter (LADDR)
122  *
123  * Must always be zero (0) as we don't implement any multi-casting.
124  * This really should look like FADR, but we do it so it's just.
125  *
126  u_long      logical_address_1;    /* LADDR */
127  u_long      logical_address_2;    /* LADDR */
128  *
129  *
130  * Receive Descriptor Ring Pointer (RDRA)
131  *
132  * The receive ring address must be quadword aligned (i.e., bits[2:0] = 0)
133  *
134  u_short      low_receive_ring_address;
135  unsigned     receive_ring_length:3; /* expressed as a power of 2 */
136  unsigned     reserved_2:5;
137  unsigned     high_receive_ring_address:8;
138  *
139  *
140  * Transmit Descriptor Ring Pointer (TDRA)
141  *
142  * The transmit ring address must be quadword aligned (i.e., bits[2:0] = 0)
143  *
144  u_short      low_transmit_ring_address;
145  unsigned     transmit_ring_length:3; /* expressed as a power of 2 */
146  unsigned     reserved_3:5;
147  unsigned     high_transmit_ring_address:8;
148  } INIT_BLOCK;
149  *
150  *
151  * Receive message descriptor structure definition
152  *
153  * This structure must be quadword aligned in system memory.
154  *
155  typedef struct receive_message_descriptor {
156  /* Receive Message Descriptor 0 (RMD0) */
157  u_short      low_buffer_address;    /* LADDR */
158  *
159  /* Receive Message Descriptor 1 (RMD1) */
160  unsigned     error_summary:1;        /* ONE HOST=0, LANCE=1 */
161  unsigned     framing_error:1;        /* FRM == Logical OR of *_errors */
162  unsigned     overflow_error:1;       /* OFLD */
163  unsigned     crc_error:1;           /* CRC */
164  unsigned     buffer_error:1;        /* BUFF */
165  unsigned     start_of_packet:1;     /* STP */
166  unsigned     end_of_packet:1;       /* EOP */
167  unsigned     high_buffer_address:8;  /* RADDR */
168  *
169  /* Receive Message Descriptor 2 (RMD2) */
170  unsigned     must_be_zero:4;
171  unsigned     buffer_byte_count:12;   /* BCNT: two's complement number */
172  *
173  /* Receive Message Descriptor 3 (RMD3) */
174  unsigned     must_be_zero:4;
175  unsigned     message_byte_count:12;  /* MCNT */
176  } RECV_MDE;
177  *
178  *
179  *
180  *
181  * Defined error returns from lance_transmit(). These exactly
182  * correspond to the TMD3, RMD error indications from the LANCE.
183  * Except, PACKET_OVERFLOW_ERROR which indicates that the received
184  * packet did not fit in a single receive message buffer - this is
185  * a fatal error as the code can not handle this situation.
186  *
187  *
188  * NOTE: all *_ERROR_* defines must be maintained as a set exclusive
189  * of each other. In particular, CERR has several *_ERROR_*
190  *
191  #define BUFFER_ERROR          0x000001
192  #define UNDERFLOW_ERROR      0x000002
193  #define LATE_COLLISION_ERROR  0x000004
194  #define LOST_CARRIER_ERROR   0x000008
195  #define RETRY_ERROR           0x000010
196  #define FRAMING_ERROR         0x000020
197  #define OVERFLOW_ERROR        0x000040
198  #define CRC_ERROR             0x000080
199  #define PACKET_OVERFLOW_ERROR 0x000100
200  #define LANCE_TIMEOUT_ERROR   0x002000
201  #define BUFFER_TOO_SMALL      0x004000
202  #define FAILURE_TO_WRITE_TO_LANCE_CSR 0x008000
203  #define FAILURE_TO_READ_FROM_LANCE_CSR 0x010000
204  #define TRANSMITTER_IS_OFF    0x020000
205  #define RECEIVER_IS_OFF       0x040000
206  *
207  *
208  * Transmit message descriptor structure definition
209  *
210  * This structure must be quadword aligned in system memory.
211  *
212  typedef struct transmit_message_descriptor {
213  /* Transmit Message Descriptor 0 (TMD0) */
214  u_short      low_buffer_address;    /* LADDR */
215  *
216  /* Transmit Message Descriptor 1 (TMD1) */
217  u_short      low_buffer_address;    /* ONE HOST=0, LANCE=1 */
218  u_short      error_summary:1;        /* FRM == Logical OR of *_errors */
219  unsigned     reserved_as_zero_1:1;   /* RES */
220  unsigned     multiple_retries:1;     /* MRE */
221  unsigned     one_retry:1;            /* ONE */
222  unsigned     deferred:1;             /* DEF */
223  unsigned     start_of_packet:1;     /* STP */
224  unsigned     end_of_packet:1;       /* EOP */
225  unsigned     high_buffer_address:8;  /* RADDR */
226  *
227  /* Transmit Message Descriptor 2 (TMD2) */
228  unsigned     must_be_zero:4;
229  unsigned     buffer_byte_count:12;   /* BCNT: two's complement number */
230  *
231  /* Transmit Message Descriptor 3 (TMD3) */
232  unsigned     buffer_error:1;         /* BUFF == T-off */
233  unsigned     underflow_error:1;      /* UFLO == T-off */
234  unsigned     reserved_as_zero_2:1;   /* RES */
235  unsigned     late_collision_error:1; /* LCOL */
236  unsigned     lost_carrier_error:1;   /* LCCR */
237  unsigned     retry_error:1;          /* RETRY */
238  unsigned     time_domain_reflectometry:10; /* TDR */
239  } TRANSM_MDE;
```

Copyright 1989 Logic Modeling Systems	HEADER FILE	DATE	PAGE #
	include/lance.h	5/23/89	3/30
		TIME	1:20:18 pm

LINE #	HEADER TEXT
241	/*
242	/* The number of buffers that are allocated for reception and transmission
243	/* must be equal for the TTTT-like protocol that ... ed to work properly.
244	/* I have specified two sets of #defines in the (un...)likely event that we ever
245	/* have to change the upper level protocol to something other than TTTT-like.
246	/*
247	/* The maximum number of buffers that the LANCE can handle in it's receive
248	/* and transmit descriptor rings is 128. (i.e., MAX_*_BUFFERS must be <= 128).
249	*/
250	#define MAX_RECEIVE_BUFFERS 64
251	#define POWER_OF_2_MAX_RECEIVE_BUFFERS 6
252	/*
253	#define MAX_TRANSMIT_BUFFERS 64
254	#define POWER_OF_2_MAX_TRANSMIT_BUFFERS 6
255	/*
256	/* This buffer allocations are used during network boot.
257	/* it is advantageous to have as few buffers as possible.
258	/* there is more RAM available to download
259	*/
260	#define MAX_RECEIVE_BT_BUFFERS 4
261	#define POWER_OF_2_RECEIVE_BT_BUFFERS 2
262	/*
263	#define MAX_TRANSMIT_BT_BUFFERS 4
264	#define POWER_OF_2_TRANSMIT_BT_BUFFERS 2
265	/*
266	/* FIXIT: buffer sizes */
267	/*
268	/* The size of the buffers needs to be adjusted to something more reasonable
269	/* based upon the host machine's Ethernet support and our upper level protocol.
270	/* Again, these must be the same size and have been specified as a pair per above.
271	/*
272	/* Ethernet specifies the minimum packet size at 64 bytes.
273	/* Remember to pad out the DATA area to 46 bytes if it is not that big.
274	/*
275	/* FIXIT: buffer sizes */
276	/*
277	/* The buffer size must contain space for the Ethernet IP and UDP headers.
278	/* This must be carefully coordinated with the client #defines and code.
279	/*
280	/* The LANCE requires that the minimum receive buffer size be >= 64 bytes.
281	/* The LANCE requires that the minimum transmit buffer size be >= 64 bytes
282	/* when data-chaining is NOT used and >= 100 bytes when it is.
283	/*
284	#define MAX_RECEIVE_BUFFER_SIZE 1518
285	#define MAX_TRANSMIT_BUFFER_SIZE 1518
286	/*
287	extern u_short get_lance_ready_to_go();
288	extern u_short start_lance();
289	extern u_short read_lance_ok();
290	extern u_short write_lance_ok();
291	/*
292	extern void shutdown_lance();
293	extern void shutdown_and_reset_lance();
294	/*

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lm_diags.h	DATE 5/23/89	PAGE # 1/31
LINE #		HEADER TEXT		
1		/* SOCI ID: lm_diags.h rev 3.1, 4/24/89 at 07:33:40 */		
2		/* Attributes fields */		
3		#define LM_DIAG_log_results 0x0001 /* log test results */		
4		#define LM_DIAG_dflt_msg_off 0x0002 /* default to messages off */		
5		#define LM_DIAG_no_select 0x0004 /* menu item will not be selectable */		
6		#define LM_DIAG_no_display 0x0008 /* menu item will not be displayed */		
7		#define LM_DIAG_no_execute 0x0010 /* menu item will not be executed		
8		when "all" is selected */		
9		#define LM_DIAG_wizard_mode 0x0020 /* menu item will display/execute		
10		in "wizard" mode only */		
11		#define LM_DIAG_no_repeat 0x0040 /* menu item cannot be repeated */		
12		#define LM_DIAG_automatic_quit 0x0080 /* quits after see selection */		
13		#define LM_DIAG_no_summary 0x0100 /* menu item will not display summary */		
14		#define LM_DIAG_acceptance 0x0200 /* execute item in acceptance test */		
15		#define LM_DIAG_no_banner 0x0400 /* don't print usual banner */		
16		#define LM_DIAG_menu_banner 0x0800 /* print lead banner only if running		
17		"all" or "acceptance" */		
18		#define LM_DIAG_another_menu (LM_DIAG_no_repeat		
19		LM_DIAG_no_summary		
20		LM_DIAG_menu_banner)		
21		#define LM_DIAG_dlog_routine (LM_DIAG_dflt_msg_off		
22		LM_DIAG_log_results)		
23		#define LM_DIAG_disable (LM_DIAG_no_execute		
24		LM_DIAG_no_select)		
25		#define LM_DIAG_menu_utility (LM_DIAG_no_repeat		
26		LM_DIAG_no_summary		
27		LM_DIAG_wizard_mode)		
28		#define LM_DIAG_utility_menu (LM_DIAG_another_menu		
29		LM_DIAG_utility)		
30		#define LM_DIAG_utility_repeat (LM_DIAG_utility LM_DIAG_no_repeat)		
31		/* Status bits */		
32		#define LM_DIAG_mail 0x0000 /* Initial value for status field */		
33		#define LM_DIAG_run 0x0001 /* menu item has been run */		
34		#define LM_DIAG_fail 0x0002 /* menu item didn't return "success" */		
35		#define LM_DIAG_stop_test 0x0004 /* menu item requests no further		
36		tests be run for this menu.		
37		this is a "post" mechanism		
38		while running "all" */		
39		/* Status word/bit setting macros */		
40		#define TEST_RUN(menu_ptr) ((LM_DIAG_MENU *)\		
41		menu_ptr->menu_items[(LM_DIAG_MENU *)\		
42		menu_ptr->current_selection].status = LM_DIAG_run		
43		#define TEST_FAILED(menu_ptr) ((LM_DIAG_MENU *)\		
44		menu_ptr->menu_items[(LM_DIAG_MENU *)\		
45		menu_ptr->current_selection].status = LM_DIAG_fail		
46		#define STOP_TEST(menu_ptr) ((LM_DIAG_MENU *)\		
47		menu_ptr->menu_items[(LM_DIAG_MENU *)\		
48		menu_ptr->current_selection].status = LM_DIAG_stop_test		
49		typedef struct {		
50		char *selection; /* the character string for selecting the item */		
51		char *menu_text; /* a description of the menu item */		
52		int (*action_routine)(); /* the address of the routine to run */		
53		long attributes; /* see attributes bits above */		
54		long status; /* return status, see above */		
55		char *user_data; /* pointer to user data, passed to menu routine */		
56		} LM_DIAG_MENU_ITEM;		
57		typedef struct {		
58		char *title; /* the title of the menu */		
59		short number_of_items; /* number of items in the menu */		
60		short current_selection; /* index into menu items */		
61		LM_DIAG_MENU_ITEM *menu_items; /* address of the menu structure array */		
62		} LM_DIAG_MENU;		
63		#define MAX_STR 255		
64		#define MAX_DEPTH 20		
65		/* Global counters */		
66		extern long total_cycles[];		
67		extern long total_tests[];		
68		extern long total_errors[];		
69		extern long total_error_msgs[];		
70		extern long total_warnings[];		
71		extern long total_messages[];		
72		/* External function definitions */		
73		char lm_get_key();		
74		char *lm_get_line();		
75		/* Network diag command tokens and useful definitions */		
76		#define BS 8		
77		#define LF 10		
78		#define CR 13		
79		#define DEL 127		
80		#define ctrl(c) ('c' & 0x1f)		
81		#define UPPER_CASE(c) (((c) > 'a') && ((c) <= 'z')) ? (c) - 'a' + 'A' : (c)		
82		#define INTR_CHARACTER ctrl(Z)		
83		/* Note that all tokens are even-numbered so that the core modeler can		
84		complain properly.		
85		/*		
86		#define LM_DIAG_SELECT 0x102		
87		#define LM_DIAG_GOTIT 0x104 /* Acknowledge token */		
88		#define LM_DIAG_RESULT 0x106		
89		#define LM_DIAG_ITEM 0x108		
90		#define LM_DIAG_TITLE 0x10a		
91		#define LM_DIAG_WHITE 0x10c		
92		#define LM_DIAG_ERROR 0x10e		
93		#define LM_DIAG_HELP 0x110		
94		#define LM_DIAG_NULL 0x112		
95		#define LM_DIAG_WARN 0x114		
96		#define LM_DIAG_PESS 0x116		
97		#define LM_DIAG_START 0x118		
98		#define LM_DIAG_END 0x11a		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lm_diags.h	DATE 5/23/89	PAGE # 2/32
LINE #		HEADER TEXT		
121	#define LM_DIAG_CINCE	0x11c /* request next line */		
122	#define LM_DIAG_GETKEY	0x11e		
123	#define LM_DIAG_GETLINE	0x120		
124	#define LM_DIAG_CHECKKEY	0x122		
125	#define LM_DIAG_GETTER	0x124		
126	#define LM_DIAG_MOVEKEY	0x126		
127	#define LM_DIAG_TEST_FAILED	0x128		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lm_rd_wr.h	DATE 5/23/89	PAGE # 1/33
LINE #		HEADER TEXT		
1		/* SOCC ID: lm_rd_wr.h rev 3.1, 4/24/89 at 07:33:43 */		
2		/*		
3		/* Interf. to read/write from modular.		
4		/* In Re.		
5		/* A maximum of 512 bytes can be read or written at a time.		
6		/*		
7		/*		
8		/*		
9		/*		
10		/* Types of accessible memories		
11		/*		
12		/*		
13		#define LM_CPRAM_MEMORY (u_long)0		
14		#define LM_NVRAM_MEMORY (u_long)1		
15		#define LM_EPRAM_MEMORY (u_long)2		
16		#define LM_IDPRAM_MEMORY (u_long)3		
17		#define LM_IDPRAM_MEMORY (u_long)4		
18		#define LM_IDPRAM_MEMORY (u_long)5		
19		#define LM_IDPRAM_MEMORY (u_long)6		
20		#define LM_IDPRAM_MEMORY (u_long)7		
21		#define LM_IDPRAM_MEMORY (u_long)8		
22		#define LM_IDPRAM_MEMORY (u_long)9		
23		#define LM_IDPRAM_MEMORY (u_long)10		
24		#define LM_IDPRAM_MEMORY (u_long)11		
25		#define LM_IDPRAM_MEMORY (u_long)12		
26		#define LM_IDPRAM_MEMORY (u_long)13		
27		#define LM_IDPRAM_MEMORY (u_long)14		
28		#define LM_IDPRAM_MEMORY (u_long)15		
29		#define LM_IDPRAM_MEMORY (u_long)16		
30		#define LM_IDPRAM_MEMORY (u_long)17		
31		#define LM_IDPRAM_MEMORY (u_long)18		
32		#define LM_IDPRAM_MEMORY (u_long)19		
33		#define LM_IDPRAM_MEMORY (u_long)20		
34		#define LM_IDPRAM_MEMORY (u_long)21		
35		#define LM_IDPRAM_MEMORY (u_long)22		
36		#define LM_IDPRAM_MEMORY (u_long)23		
37		#define LM_IDPRAM_MEMORY (u_long)24		
38		#define LM_IDPRAM_MEMORY (u_long)25		
39		#define LM_IDPRAM_MEMORY (u_long)26		
40		#define LM_IDPRAM_MEMORY (u_long)27		
41		#define LM_IDPRAM_MEMORY (u_long)28		
42		#define LM_IDPRAM_MEMORY (u_long)29		
43		#define LM_IDPRAM_MEMORY (u_long)30		
44		#define LM_IDPRAM_MEMORY (u_long)31		
45		#define LM_IDPRAM_MEMORY (u_long)32		
46		#define LM_IDPRAM_MEMORY (u_long)33		
47		#define LM_IDPRAM_MEMORY (u_long)34		
48		#define LM_IDPRAM_MEMORY (u_long)35		
49		#define LM_IDPRAM_MEMORY (u_long)36		
50		#define LM_IDPRAM_MEMORY (u_long)37		
51		#define LM_IDPRAM_MEMORY (u_long)38		
52		#define LM_IDPRAM_MEMORY (u_long)39		
53		#define LM_IDPRAM_MEMORY (u_long)40		
54		#define LM_IDPRAM_MEMORY (u_long)41		
55		#define LM_IDPRAM_MEMORY (u_long)42		
56		#define LM_IDPRAM_MEMORY (u_long)43		
57		#define LM_IDPRAM_MEMORY (u_long)44		
58		#define LM_IDPRAM_MEMORY (u_long)45		
59		#define LM_IDPRAM_MEMORY (u_long)46		
60		#define LM_IDPRAM_MEMORY (u_long)47		
61		#define LM_IDPRAM_MEMORY (u_long)48		
62		#define LM_IDPRAM_MEMORY (u_long)49		
63		#define LM_IDPRAM_MEMORY (u_long)50		
64		#define LM_IDPRAM_MEMORY (u_long)51		
65		#define LM_IDPRAM_MEMORY (u_long)52		
66		#define LM_IDPRAM_MEMORY (u_long)53		
67		#define LM_IDPRAM_MEMORY (u_long)54		
68		#define LM_IDPRAM_MEMORY (u_long)55		
69		#define LM_IDPRAM_MEMORY (u_long)56		
70		#define LM_IDPRAM_MEMORY (u_long)57		
71		#define LM_IDPRAM_MEMORY (u_long)58		
72		#define LM_IDPRAM_MEMORY (u_long)59		
73		#define LM_IDPRAM_MEMORY (u_long)60		
74		#define LM_IDPRAM_MEMORY (u_long)61		
75		#define LM_IDPRAM_MEMORY (u_long)62		
76		#define LM_IDPRAM_MEMORY (u_long)63		
77		#define LM_IDPRAM_MEMORY (u_long)64		
78		#define LM_IDPRAM_MEMORY (u_long)65		
79		#define LM_IDPRAM_MEMORY (u_long)66		
80		#define LM_IDPRAM_MEMORY (u_long)67		
81		#define LM_IDPRAM_MEMORY (u_long)68		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lm_sfi.h	DATE 5/23/89	PAGE # 1/34
LINE #		HEADER TEXT		
1	2	/* SOCS ID: lm_sfi.h rev 3.1, 4/24/89 at 07:33:47 */		
3	4	/* BEGIN LM DEFINITION */		
5	6	/* Return codes for all SFI functions */		
7	8	#define LM_SUCCESS 1		
9	10	#define LM_WARNING 0		
11	12	#define LM_ERROR -1		
13	14	#define LM_TRUE 1		
15	16	#define LM_FALSE 0		
17	18	/* Constants for lm_status_definition */		
19	20	#define LM_END_OF_TEXT (long)0		
21	22	/* Bit 25 set, in C: 0x00000000 */		
23	24	#define LM_INVALID_PIN_ID (long)268435456		
25	26	/* Constants for lm_release */		
27	28	#define LM_ALL_DEFINITIONS (long)-2		
29	30	#define LM_ALL_INSTANCES (long)-3		
31	32	#define LM_ALL_FAULTS (long)-4		
33	34	/* Constants for lm_verify_simulator */		
35	36	#define LM_FAULT_SIMULATOR (long)16777216		
37	38	#define LM_LOGIC_SIMULATOR (long)16777217		
39	40	/* Attributes for lm_set_logic_level_map */		
41	42	#define LM_LOGIC_ZERO (long)2		
43	44	#define LM_LOGIC_SOFT_ZERO (long)3		
45	46	#define LM_LOGIC_ONT (long)4		
47	48	#define LM_LOGIC_SOFT_ONT (long)5		
49	50	#define LM_LOGIC_FLOAT (long)6		
51	52	#define LM_LOGIC_UNKNOWN (long)7		
53	54	/* Constants for lm_set_pin_map */		
55	56	#define LM_INPUT (long)9		
57	58	#define LM_OUTPUT (long)10		
59	60	#define LM_IO (long)11		
61	62	#define LM_IO_AS_INPUT (long)12		
63	64	#define LM_IO_AS_OUTPUT (long)13		
65	66	#define LM_DONT_CARE (long)14		
67	68	/* Constants for lm_pattern_history */		
69	70	#define LM_KEEP_PATTERNS (long)15		
71	72	#define LM_DONT_KEEP_PATTERNS (long)16		
73	74	/* Constants for lm_inquire_modeler */		
75	76	#define LM_TOTAL_MODELER_MEMORY (long)18		
77	78	#define LM_TOTAL_PATTERNS (long)19		
79	80	#define LM_AVAILABLE_MODELER_MEMORY (long)20		
81	82	#define LM_AVAILABLE_PATTERNS (long)21		
83	84	#define LM_NUMBER_OF_USERS (long)22		
85	86	#define LM_SWITCH_USER_AM_I (long)23		
87	88	#define LM_NUMBER_OF_LANES (long)24		
89	90	#define LM_NUMBER_OF_SLOTS_PER_LANE (long)25		
91	92	#define LM_NUMBER_OF_PACS (long)26		
93	94	#define LM_NUMBER_OF_PINS (long)27		
95	96	#define LM_NUMBER_OF_PELS (long)28		
97	98	#define LM_NUMBER_OF_DAPS (long)29		
99	100	#define LM_NUMBER_OF_ACTIVE_INSTANCES (long)30		
101	102	#define LM_NUMBER_OF_ACTIVE_FAULTS (long)31		
103	104	#define LM_NUMBER_OF_PUBLIC_DEVICES (long)32		
105	106	#define LM_NUMBER_OF_PRIVATE_DEVICES (long)33		
107	108	#define LM_SOFTWARE_REVISION_NUMBER (long)34		
109	110	/* Constants for lm_inquire_user */		
111	112	#define LM_NUMBER_OF_PATTERNS (long)35		
113	114	#define LM_SECONDS_SINCE_LAST_RESPONSE (long)36		
115	116	/* Constants for lm_inquire_lane */		
117	118	#define LM_IS_LANE_USABLE (long)37		
119	120	#define LM_NUMBER_OF_POPULATED_PELS (long)38		
121	122	/* Constants for lm_inquire_pac */		
123	124	#define LM_PAC_MEMORY_SIZE (long)39		
125	126	/* Constants for lm_inquire_pelt */		
127	128	#define LM_PELS_PEL_EXIST (long)40		
129	130	#define LM_IS_DAB_PRESENT (long)41		
131	132	/* Constants for lm_inquire_dab */		
133	134	#define LM_DAB_TYPE (long)42		
135	136	#define LM_DEVICE_STATUS (long)43		
137	138	#define LM_DAB_REVISION (long)44		
139	140	#define LM_DAB_MANUFACTURER (long)45		
141	142	#define LM_DAB_MODEL (long)46		
143	144	#define LM_DAB_REVISION (long)47		
145	146	#define LM_PRIVATE (long)48		
147	148	#define LM_PUBLIC (long)49		
149	150	/* Constants for lm_inquire_instance */		
151	152	#define LM_NUMBER_OF_FAULTS (long)50		
153	154	#define LM_ASSOCIATED_DEFINITION_ID (long)51		
155	156	/* Constants for lm_inquire_fault */		
157	158	#define LM_ASSOCIATED_INSTANCE_ID (long)52		
159	160	/* Attributes for lm_get_message */		
161	162	#define LM_NO_MORE_MESSAGES (long)53		
163	164	/* Entities for lm_timing_measurement */		
165	166	#define LM_TIMING_ON (long)54		
167	168	#define LM_TIMING_OFF (long)55		
169	170	#define LM_NO_FILE (char *)NULL		
171	172	/* Entities for lm_eval_control */		
173	174	#define LM_NUMBER_OF_SAMPLES (long)56		
175	176	#define LM_NUMBER_OF_EVALUATIONS (long)57		
177	178	#define LM_EVERY_EVALUATION (long)-1		
179	180	/* Entities for lm_inquire_device_pins */		
181	182	#define LM_POWER (long)58		
183	184	#define LM_CLOCKED (long)59		
185	186	#define LM_NO_CONNECT (long)60		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lm_sfi.h	DATE 5/23/89 TIME 1:20:19 pm	PAGE # 2/35
LINE #	HEADER TEXT			
121	#define LM_DATA_PIN (long)61			
122	#define LM_EVAL_PIN (long)62			
123	#define LM_STORE_PIN (long)63			
124	#define LM_EDGE_RISE_PIN (long)64			
125	#define LM_EDGE_FALL_PIN (long)65			
126	#define LM_NO_NOISE_PINS (char *)NULL			
127	/* additional entities for lm_inquire_modeler() */			
128	#define LM_EPOCH_ADAPTER_INSERTIONS (long)66			
129	#define LM_EPOCH_LONGEST_SEQUENCE (long)67			
130	#define LM_EPOCH_PATTERNS_ADDED_HIGH (long)68			
131	#define LM_EPOCH_PATTERNS_ADDED_LOW (long)69			
132	#define LM_EPOCH_DEFINITIONS (long)70			
133	#define LM_EPOCH_INSTANCES (long)71			
134	#define LM_EPOCH_FAULTS (long)72			
135	/* entities for lm_get_pin_change() */			
136	#define LM_TABLE_DELAY (long)73			
137	#define LM_MEASURED_DELAY (long)74			
138	#define LM_COMPOSITE_DELAY (long)75			
139	#define LM_UNSPECIFIED_DELAY (long)76			
140	/* additional entities for lm_inquire_instance() */			
141	#define LM_PATTERN_FREQUENCY (long)77			
142	/* entities for lm_set_simulation_tick() */			
143	#define LM_PICOSECONDS (long)78			
144	#define LM_FEMTOSECONDS (long)79			
145	#define LM_NANOSECONDS (long)80			
146	#define LM_MICROSECONDS (long)81			
147	/* entities for lm_log_test_vector() */			
148	#define LM_LOGGING_ON (long)82			
149	#define LM_LOGGING_OFF (long)83			
150	/* entities for lm_loop_patterns() */			
151	#define LM_LOOP_ON (long)85			
152	#define LM_LOOP_OFF (long)86			
153	/* Additional attributes to lm_inquire_modeler() */			
154	#define LM_PASSWORD_CHECK (long)87			
155	/* entities for lm_password() */			
156	#define LM_ASSIGN (long)88			
157	#define LM_ENTER (long)89			
158	/* Additional attributes to lm_inquire_user() */			
159	#define LM_NUMBER_OF_PATTERNS_ALLOCATED (long)90			
160	#define LM_GET_VERSION_STRING_ADDR (long)91			
161	/* Maximum legal attribute value for all lm calls */			
162	#define LM_MAXIMUM_ATTRIBUTE_VALUE (long)92			
163	/* END LM DEFINITION */			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsbsp.h	DATE 5/23/89	PAGE # 1/36
LINE #		HEADER TEXT		
1		/* SCCS Id: lmsbsp.h rev 1.1. 4/24/89 at 07:31:50 */		
2		/* NOTE: this must match CPU_RAM in cpa.h */		
3		#define CPU_INIT_RAM 0x0000		
4		/* Hypothetical Workspace Address */		
5		#define HX_CS_ADD (CPU_INIT_RAM + 0x4000)		
6		/*		
7		Vrtx Configuration Defines		
8		/*		
9		#define HX_MAX 0 /* Ready Systems highest comp. no. */		
10		#define HX_MAX 0 /* Task highest comp. no. */		
11		#define VRTX_WK_SPACE (CPU_INIT_RAM + 0x5000) /* VRTX workspace address */		
12		#define VRTX_WK_SIZE 0x1000 /* VRTX workspace size */		
13		#define STX_STX 0x2000 /* System stack size */		
14		#define ISE_STX 0x2000 /* Interrupt stack size */		
15		#define CB_COUNT 10 /* VRTX CB number MAX-4 */		
16		#define WC_DIS_LEVEL 7 /* Component disable level */		
17		#define HX_STX 0x2000 /* User stack size */		
18		#define TASK_COUNT 7 /* Number of VRTX tasks */		
19		#define TRP_VRTX 0x0 /* VRTX Trp Number */		
20		/*		
21		Ktscope Configuration Defines		
22		/*		
23		#define KTS_WK_SPACE (CPU_INIT_RAM + VRTX_WK_SPACE + VRTX_WK_SIZE) /* Ktscope workspace address */		
24		#define KTS_WK_SIZE 0x1000 /* Ktscope workspace size */		
25		#define WC_DIS_LEVEL 0 /* Component disable level */		
26		#define KTS_PRI 20 /* Ktscope task priority level */		
27		#define KTS_TID 1 /* Ktscope task ID number */		
28		/*		
29		Address of RAM location for System Configuration Tables		
30		/*		
31		#define STX_TABLE (CPU_INIT_RAM + 0x1000)		
32		/*		
33		Ktscope Component Addresses and Defines		
34		/*		
35		#define KTS_BASE (CPU_INIT_RAM + KTS_WK_SPACE + KTS_WK_SIZE) /* Ktscope physical address */		
36		#define VRTX_BASE (CPU_INIT_RAM + 0x02000) /* VRTX physical address */		
37		#define KTS_ENTRY 0x0 /* Ktscope entry point (offset from the physical address) */		
38		/*		
39		BSP Data Structures		
40		/*		
41		Component Vector Table		
42		/*		
43		typedef struct		
44		{		
45		char hr_max;		
46		char wr_max;		
47		char read[6];		
48		char read[8];		
49		long comp_code;		
50		long comp_work;		
51		long comp_code;		
52		long comp_work;		
53		} CVT_TBL;		
54		/*		
55		Ktscope Configuration Tables		
56		/*		
57		typedef struct		
58		{		
59		char *cb_short;		
60		char *cb_aline;		
61		char *cb_backspc;		
62		char *cb_newline;		
63		char *cb_line_rept;		
64		char *cb_up_arrow;		
65		char *cb_down_arrow;		
66		char *cb_repeat;		
67		} DS_SPEC;		
68		/*		
69		typedef struct		
70		{		
71		long cb_gid;		
72		long cb_gsize;		
73		short cb_inpt_ll;		
74		short cb_hstck;		
75		long cb_aline;		
76		long cb_sys_tab;		
77		short cb_psm_chal;		
78		char cb_tspole;		
79		char cb_xss;		
80		char cb_xoff;		
81		char cb_xrft;		
82		long cb_trury;		
83		long cb_in_filter;		
84		long cb_out_filter;		
85		long cb_pstcl;		
86		long cb_ppstcl;		
87		long cb_pstcl;		
88		long cb_ppstcl;		
89		long cb_pstcl;		
90		long cb_ppstcl;		
91		long cb_pstcl;		
92		long cb_ppstcl;		
93		long cb_pstcl;		
94		long cb_ppstcl;		
95		long cb_pstcl;		
96		long cb_ppstcl;		
97		long cb_pstcl;		
98		long cb_ppstcl;		
99		long cb_pstcl;		
100		long cb_ppstcl;		
101		long cb_pstcl;		
102		long cb_ppstcl;		
103		long cb_pstcl;		
104		long cb_ppstcl;		
105		long cb_pstcl;		
106		long cb_ppstcl;		
107		long cb_pstcl;		
108		long cb_ppstcl;		
109		long cb_pstcl;		
110		long cb_ppstcl;		
111		long cb_pstcl;		
112		long cb_ppstcl;		
113		long cb_pstcl;		
114		long cb_ppstcl;		
115		long cb_pstcl;		
116		long cb_ppstcl;		
117		long cb_pstcl;		
118		long cb_ppstcl;		
119		long cb_pstcl;		
120		long cb_ppstcl;		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsbsp.h	DATE 5/23/89	PAGE # 2/37
LINE #		HEADER TEXT		
121		typedef struct		
122		{		
123		long db_vrtx_c/		
124		long db_rt_v/		
125		long db_vt_size/		
126		short db_illegal/		
127		long db_trap_c/		
128		short db_comp_dis/		
129		short db_priority/		
130		short db_tid/		
131		long db_trap/		
132		long db_trap_c/		
133		long db_vrtx_d/		
134		long db_trap_vrtx/		
135		long read[3]/		
136		}		
137		RTCT_CTRL/		
138		/*----- Vrtx Configuration Table -----*/		
139		typedef struct		
140		{		
141		long v_vrtx_c/		
142		long v_vt_size/		
143		short sys_vt_size/		
144		short lar_vt_size/		
145		short cb_count/		
146		short read/		
147		long read/		
148		short c_dis_lev/		
149		short war_vt_size/		
150		long read/		
151		short war_vt_count/		
152		short read/		
153		long tx_rdy/		
154		long tx_rdy/		
155		long tx_rdy/		
156		long tx_rdy/		
157		long tx_rdy/		
158		long tx_rdy/		
159		long tx_rdy/		
160		long tx_rdy/		
161		long tx_rdy/		
162		long tx_rdy/		
163		long tx_rdy/		
164		long tx_rdy/		
165		long tx_rdy/		
166		long tx_rdy/		
167		long tx_rdy/		
168		long tx_rdy/		
169		long tx_rdy/		
170		long tx_rdy/		
171		long tx_rdy/		
172		long tx_rdy/		
173		long tx_rdy/		
174		long tx_rdy/		
175		long tx_rdy/		
176		long tx_rdy/		
177		long tx_rdy/		
178		long tx_rdy/		
179		long tx_rdy/		
180		long tx_rdy/		
181		long tx_rdy/		
182		long tx_rdy/		
183		long tx_rdy/		
184		long tx_rdy/		
185		long tx_rdy/		
186		long tx_rdy/		
187		long tx_rdy/		
188		long tx_rdy/		
189		long tx_rdy/		
190		long tx_rdy/		
191		long tx_rdy/		
192		long tx_rdy/		
193		long tx_rdy/		
194		long tx_rdy/		
195		long tx_rdy/		
196		long tx_rdy/		
197		long tx_rdy/		
198		long tx_rdy/		
199		long tx_rdy/		
200		long tx_rdy/		
201		long tx_rdy/		
202		long tx_rdy/		
203		long tx_rdy/		
204		long tx_rdy/		
205		long tx_rdy/		
206		long tx_rdy/		
207		long tx_rdy/		
208		long tx_rdy/		
209		long tx_rdy/		
210		long tx_rdy/		
211		long tx_rdy/		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsrver.h	DATE 5/23/89	PAGE # 1/38
LINE #		HEADER TEXT		
1		/* SOCK_FD: lmsrver.h rev 3.1, 4/24/89 at 07:33:53 */		
2		extern char *lm_realloc();		
3		extern char *lm_malloc();		
4		extern char *lm_free();		
5		#define DREALLOC lm_realloc		
6		#define DFREE lm_free		
7		#define DREALLOC lm_malloc		
8		#define DREALLOC lm_malloc		
9		#define SOFTWARE_REVISION_NUMBER 0x01000000		
10		#define LAST_COMPATIBLE_CMC_VERSION 0x01000000		
11		#define MAX_STRING_LENGTH 256		
12		#define MAXINT (((unsigned)0) >> 1)		
13		#define MAX_USER_COUNT 64		
14		#define MAXNSC (16*1024)		
15		#define MAXNSC_INCR (2*1024)		
16		#define MAX_FUNCTION 100		
17		#define MAX_LASH_COUNT 4		
18		#define MAX_SLOT_COUNT 4		
19		#define MAX_PAM_COUNT 4		
20		#define MAX_MAGIC_COUNT 5		
21		#define TYPE_BIT_SHIFT 4		
22		#define MAX_UNIT_COUNT 16		
23		#define MAX_PIN_PER_UNIT 80		
24		#define MAX_PIN_COUNT (MAX_UNIT_COUNT * MAX_PIN_PER_UNIT)		
25		#define MAX_BIT_PER_WORD (sizeof(int)*8)		
26		#define INST_TABLE_SIZE 16		
27		#define INST_TABLE_INCR 16		
28		#define DEF_TABLE_SIZE 8		
29		#define DEF_TABLE_INCR 8		
30		#define MAX_EVAL_CHANGES 320		
31		#define EVAL_EVENT 0		
32		#define STORE_EVENT 1		
33		#define NO_TRANSITION 0		
34		#define RISE_TRANSITION 1		
35		#define FALL_TRANSITION 2		
36		/* Possible values for pin values passed between host API and modules */		
37		#define LOGIC_0 0x1		
38		#define LOGIC_1 0x2		
39		#define LOGIC_10 0x4		
40		#define LOGIC_21 0x8		
41		#define LOGIC_U 0x10		
42		#define LOGIC_20 0x20		
43		#define LOGIC_31 0x40		
44		#define LOGIC_X 0x60		
45		#define LOGIC_Z 0x80		
46		#define LOGIC_MAX_STATE LOGIC_31		
47		#define EDGE_RISE STORE + 1		
48		#define EDGE_FALL STORE + 2		
49		/* divide the pattern count by 32 for debugging purposes */		
50		#define PAM12K_PTN_COUNT (128 * 1024)		
51		#define PAM12K_PTN_COUNT (512 * 1024)		
52		#define PAM12K_PTN_COUNT (2 * 1024 * 1024)		
53		#define PAM12K_PTN_COUNT (128 * 1024 / 32)		
54		#define PAM12K_PTN_COUNT (512 * 1024 / 32)		
55		#define PAM12K_PTN_COUNT (2 * 1024 * 1024 / 32)		
56		#define PAM12K_PTN_COUNT 3		
57		#define PAM12K_PTN_COUNT 4		
58		#define PAM12K_PTN_COUNT 8		
59		#define MAX_PAM_CONFIG 34		
60		#define PAM12K_WEIGHT 1		
61		#define PAM12K_WEIGHT 5		
62		#define PAM12K_WEIGHT 21		
63		#define BLOCK_START_MASK 0xffff0000		
64		#define BLOCK_NUMBER_SHIFT 16		
65		#define BLOCK_NUMBER_MASK 0xffff		
66		#define PTN_ADDR_INC 4		
67		#define LINK_TABLE_ADDR_INC 32 * 1024		
68		#define MAX_LINK_TABLE_COUNT 256		
69		#define BLOCK_ADDR_INC (PTN_PER_BLOCK * PTN_ADDR_INC)		
70		#define BRANCH_LATENCY 3		
71		#define FB_LATENCY 3		
72		#define FREE_BLOCK_FLAG 0xffff		
73		#define LAST_PATTERN_IN_BLOCK_MASK 0x1c		
74		#define FB_PTN_COUNT 512		
75		/* The following number is determined empirically */		
76		#define MAX_DWNSY_COUNT_1_BIT_PER_PIN 28		
77		#define MAX_DWNSY_COUNT_2_BIT_PER_PIN 44		
78		#define END_BLOCK_FLAG 0 /* flag in link table for last block in the pattern sequence */		
79		#define SEQ_END_LATENCY 3		
80		#define PEL_SEL_BIT_SOCKET 15		
81		#define PEL_SEL_MASK 0xffffffff		
82		#define PEL_CTL_MASK 0xffffffff		
83		#define PEL_USER_MASK 0xffffffff		
84		#define PEL_STOP_MASK 0xffffffff		
85		#define PEL_BRANCH_MASK 0xffffffff		
86		#define CTL_PTN_MASK 0xffffffff		
87		#define PEL_SEL_SHIFT 0		
88		#define PEL_CTL_SHIFT 4		
89		#define PEL_BRANCH_SHIFT 3		
90		#define PEL_CTL_WOOP 0		
91		#define PEL_CTL_LOAD_SDM_FORMAT 1		
92		#define PEL_CTL_SELECT_PEL 2		
93		#define PEL_CTL_LOAD_PTN 3		
94		#define PEL_CTL_LOAD_DATA 4		

Copyright 1989 by Logic Modeling Systems		HEADER FILE include/lmserver.h	DATE 5/23/89	PAGE # 2/39
LINE #		HEADER TEXT		
121	#define PEL_CTL_LOAD_CONTROL	5		
122	#define PEL_CTL_LOAD_BSDEN	6		
123	#define PEL_CTL_LOAD_DATA_LAST_UNIT	7		
124	/* PEL Control/Status registers */			
125	#define PEL_CTL_IN USE_LED_MASK	0x00000000		
126	#define PEL_CTL_RESET_MASK	0x00000000		
127	#define PEL_CTL_PRIVATE_PUBLIC_MASK	0x00000000		
128	#define PEL_CTL_INTERRUPT_ENABLE_MASK	0x00000000		
129	#define PEL_CTL_INITIALIZE_MASK	0x00000000		
130	#define PEL_CTL_ACTIVE_MASK	0x00000000		
131	#define PEL_CTL_PRESENT_MASK	0x00000000		
132	#define PEL_CTL_INTERRUPT_MASK	0x00000000		
133	#define PEL_CTL_PLAY_ERROR_MASK	0x00000000		
134	#define PEL_CTL_RESET_ERROR_MASK	0x00000000		
135	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
136	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
137	#define DUMMY_CTL_WORD(slotno)	((PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT) \		
138		((slotno) << PEL_CTL_SHIFT))		
139	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
140	#define BRANCH_NEVER	0		
141	#define BRANCH_ALWAYS	1		
142	#define BRANCH_FALL	2		
143	#define BRANCH_RISE	3		
144	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
145	#define FEEDBACK_BLOCK	0		
146	#define PATTERN_BLOCK	1		
147	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
148	#define CERR_TNC_CAL	0		
149	#define CERR_NO_TNC	1		
150	#define CERR_NO_PWR_FOR_PAC	2		
151	#define CERR_PWR_STAPLED_WRONG	3		
152	#define CERR_PWR_STACKED_WRONG	4		
153	#define CERR_DUPLICATE_SEGMENT	5		
154	#define CERR_MISSING_SEGMENT	6		
155	#define CERR_NO_PEL_FOR_DAB	7		
156	#define CERR_NO_PWR_FOR_DAB	8		
157	#define CERR_ILLEGAL_DUPLICATE_DEVICE	9		
158	#define CERR_DEVICE_TOO_LARGE	10		
159	#define CERR_ILLEGAL_PEL_STACKING	11		
160	/* state of some command */			
161	/* Note: modify this state of INSTANCE_INFO restores state if the last state			
162	/* is greater than 15. */			
163	/* is greater than 15. */			
164	/* is greater than 15. */			
165	#define SENT_RECV_MIRRORING	0		
166	#define SENT_RECV_PSRN_LOADED	1		
167	#define SENT_RECV_SEGMENT_LOADED	2		
168	#define SENT_RECV_SEGMENT_LOADED	3		
169	#define SENT_RECV_SEGMENT_LOADED	4		
170	#define SENT_RECV_LAST CONSISTENT SET	5		
171	#define SENT_RECV_SIM_PIN_VALUE_DATA	6		
172	#define SENT_RECV_SIM_PIN_VALUE_HIZ	7		
173	#define SENT_RECV_SIM_PIN_VALUE_LOW	8		
174	#define SENT_RECV_SIM_PIN_VALUE_SOFT	9		
175	#define SENT_RECV_LAST SAMPLE VALUE DATA	10		
176	#define SENT_RECV_LAST SAMPLE VALUE_HIZ	11		
177	#define SENT_RECV_LAST SAMPLE VALUE_LOW	12		
178	#define SENT_RECV_LAST SAMPLE VALUE_SOFT	13		
179	#define SENT_RECV_PATTERN	14		
180	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
181	#define TNC_LOCK_TIMEOUT	5		
182	/* 0x7777 */			
183	#define PWRM_PLAY_TIMEOUT	1000 /* ms */		
184	#define PWRM_COUNT_FUDGE_FACTOR	125000		
185	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
186	#define MAX_SAMPLE_RAMP_RANGE	250		
187	#define RESOLUTION_05_MS	0		
188	#define RESOLUTION_10_MS	1		
189	#define RESOLUTION_20_MS	2		
190	#define RESOLUTION_40_MS	3		
191	#define RESOLUTION_INVALID	4		
192	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
193	#define MAX_SHORT_WORD_COUNT	5		
194	#define MAX_LONG_WORD_COUNT	3		
195	/* This is the period threshold which determines what should be written to the			
196	/* CLOCK SPEED register on the PAC. */			
197	/* This is the period at which we going to play private devices. */			
198	#define PAC_FREQUENCY_THRESHOLD	1000000 /* ps */		
199	/* This is the period at which we going to play private devices. */			
200	#define PRIVATE_PATTERN_PERIOD	5000000 /* ps == 200.KHz */		
201	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
202	#define MAX_PERCENT_TOLERANCE	5		
203	#define MAX_DOTWOC	5250 /* ms */		
204	#define MIN_DOTWOC	3000 /* ms */		
205	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
206	#define DEFAULT_SAMPLE COUNT	5		
207	#define DEFAULT EVALUATION COUNT	100		
208	#define MAX_CHECK_IMPORTANT COUNT	100		
209	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
210	#define CRYPTED_PASSWORD_LENGTH	16		
211	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
212	#define NO_PULLUP_OR_PULLDOWN	0		
213	#define PULLUP	1		
214	#define PULLDOWN	2		
215	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
216	typedef struct {			
217	u_long mem_addr; /* is pattern unit */			
218	u_long board_id;			
219	}; PAM_INFO;			
220	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
221	typedef struct {			
222	char revision[REV_LENGTH + 1];			
223	char dev_type[TYPE_LENGTH + 1];			
224	char mem_id[MEM_LENGTH + 1];			
225	char model_maker[MAKER_LENGTH + 1];			
226	char model_revision[REV_LENGTH + 1];			
227	u_short insertion_count;			
228	unsigned : 9;			
229	unsigned lane_no : 3;			
230	unsigned slot_no : 4;			
231	u_long shape_bit_map; /* the DAB shape map as stored in EEPROM */			
232	}; SEGMENT_EL;			
233	/* PEL_CTL_SELECT_PEL << PEL_CTL_SHIFT */			
234	typedef struct {			
235	unsigned last_is_lane : 1;			
236	unsigned lane_no : 3;			
237	unsigned slot_no : 4;			
238	}; UNIT_LOCATION_EL;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmserver.h	DATE 5/23/89 TIME 1:20:20 pm	PAGE # 3/40
LINE #	HEADER TEXT			
241	/*			
242	word[0] --> pcb129..642			
243	word[1] --> pcb131..			
244	word[2] --> pcb133..162			
245	word[3] --> pcb135..162			
246	word[4] --> pcb137..162			
247	*/			
248	typedef struct {			
249	u_short word[MAX_SHORT_WORD_COUNT];			
250	u_short ctrl;			
251	} PTM_BITS;			
252	typedef struct {			
253	u_long word[MAX_LONG_WORD_COUNT];			
254	} PTM_BITS_LONGWORD;			
255	typedef struct dab_info {			
256	segment[MAX_SEGMENT_PER_DEVICE + 1];			
257	PTM_BITS_LONGWORD data_ptr[MAX_LANE_COUNT];			
258	act_lane_count;			
259	act_var_count;			
260	part_name[NAME_LENGTH + 1];			
261	used_as_private;			
262	segment_count;			
263	unit_count_per_lane;			
264	lane_count; /* num of lanes occupied by this dab */			
265	unit_count;			
266	lane_count[MAX_LANE_COUNT];			
267	ident_lane; /* to select lane in pattern play */			
268	data_vcc_measured; /* mv */			
269	val_programmed; /* VIL currently programmed on DAB */			
270	vth_programmed; /* VSH currently programmed on DAB */			
271	vth_programmed;			
272	vth_programmed;			
273	vth_programmed;			
274	vth_programmed;			
275	vth_programmed;			
276	vth_programmed;			
277	vth_programmed;			
278	UNIT_LOCATION_EL unit_location[MAX_UNIT_COUNT];			
279	} DAB_INFO;			
280	typedef struct {			
281	u_long board_id;			
282	} PEL_INFO;			
283	typedef struct {			
284	PTM_BITS *data;			
285	PTM_BITS *bit;			
286	PTM_BITS *unknown;			
287	PTM_BITS *soft;			
288	} FULL_VALUE;			
289	/* Edge assignments:			
290	EDGE# USED AS			
291	/*			
292	0 EDG00 edge			
293	1 falling edge of EI (edge_rise pin)			
294	2 data edge			
295	3 store edge			
296	4 rising edge of EI (edge_fall pin)			
297	5 alternate EDG00 edge			
298	6 sample edge threshold we want to program at this evaluation			
299	7 sample edge threshold for store event			
300	8 sample edge threshold for eval event			
301	*/			
302	typedef struct {			
303	PTM_BITS_LONGWORD *ident_inputs;			
304	PTM_BITS_LONGWORD *ident_outputs;			
305	PTM_BITS_LONGWORD *ident_1oe;			
306	PTM_BITS_LONGWORD *ident_EI;			
307	PTM_BITS_LONGWORD *ident_EI;			
308	PTM_BITS_LONGWORD *ident_store;			
309	u_long logical_clock_period;			
310	u_long actual_phy_clock_period;			
311	u_long sample_edge_time;			
312	u_long data_edge_time;			
313	u_long ident_edge_time;			
314	u_long ident_edge_time;			
315	u_long store_edge_time;			
316	u_long non_significant_store_edge_time;			
317	u_long edge_outtime[MAX_EDGE_COUNT + 1];			
318	u_long falling_sample_reg;			
319	u_long lane_used;			
320	u_char dab_id;			
321	u_char has_io_store;			
322	u_char a_reg;			
323	u_char b_reg;			
324	u_char source_reg;			
325	u_char val_dac_value;			
326	u_char vth_dac_value;			
327	u_char vth_dac_value;			
328	u_char vth_dac_value;			
329	u_char vth_dac_value;			
330	u_char vth_dac_value;			
331	u_char private_val_dac_value;			
332	u_char private_vth_dac_value;			
333	u_char eval_event_mode;			
334	u_char store_event_mode;			
335	u_char ri_or_rs_min_threshold; /* for early sample only */			
336	} EXTRA_DEVICE_SPEC;			
337	typedef struct {			
338	u_long max_addr; /* max addr of current block, 1 ptr past the			
339	block */			
340	u_long prev_max_addr; /* max addr of previous block */			
341	u_long last_unit_addr; /* ptr addr of the last unit in the lane */			
342	u_long new_block_addr; /* addr of the new block just allocated */			
343	} LANE_ADDR_INFO;			
344	typedef struct {			
345	long min_delay; /* M/T/N Delays */			
346	long typ_delay;			
347	long max_delay;			
348	u_char old_rev;			
349	u_char old_filtered;			
350	u_char new_rev;			
351	u_char new_filtered;			
352	u_long sin_time;			
353	short next_input_pin_index;			
354	short next_output_pin_index;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsrvr.h	DATE 5/23/89	PAGE # 4/41
LINE #		HEADER TEXT		
361	362	363	364	365
366	367	368	369	370
371	372	373	374	375
376	377	378	379	380
381	382	383	384	385
386	387	388	389	390
391	392	393	394	395
396	397	398	399	400
401	402	403	404	405
406	407	408	409	410
411	412	413	414	415
416	417	418	419	420
421	422	423	424	425
426	427	428	429	430
431	432	433	434	435
436	437	438	439	440
441	442	443	444	445
446	447	448	449	450
451	452	453	454	455
456	457	458	459	460
461	462	463	464	465
466	467	468	469	470
471	472	473	474	475
476	477	478	479	480
481	482	483	484	485
486	487	488	489	490
491	492	493	494	495
496	497	498	499	500
501	502	503	504	505
506	507	508	509	510
511	512	513	514	515
516	517	518	519	520
521	522	523	524	525
526	527	528	529	530
531	532	533	534	535
536	537	538	539	540
541	542	543	544	545
546	547	548	549	550
551	552	553	554	555
556	557	558	559	560
561	562	563	564	565
566	567	568	569	570
571	572	573	574	575
576	577	578	579	580
581	582	583	584	585
586	587	588	589	590
591	592	593	594	595
596	597	598	599	600
601	602	603	604	605
606	607	608	609	610
611	612	613	614	615
616	617	618	619	620
621	622	623	624	625
626	627	628	629	630
631	632	633	634	635
636	637	638	639	640
641	642	643	644	645
646	647	648	649	650
651	652	653	654	655
656	657	658	659	660
661	662	663	664	665
666	667	668	669	670
671	672	673	674	675
676	677	678	679	680
681	682	683	684	685
686	687	688	689	690
691	692	693	694	695
696	697	698	699	700
701	702	703	704	705
706	707	708	709	710
711	712	713	714	715
716	717	718	719	720
721	722	723	724	725
726	727	728	729	730
731	732	733	734	735
736	737	738	739	740
741	742	743	744	745
746	747	748	749	750
751	752	753	754	755
756	757	758	759	760
761	762	763	764	765
766	767	768	769	770
771	772	773	774	775
776	777	778	779	780
781	782	783	784	785
786	787	788	789	790
791	792	793	794	795
796	797	798	799	800
801	802	803	804	805
806	807	808	809	810
811	812	813	814	815
816	817	818	819	820
821	822	823	824	825
826	827	828	829	830
831	832	833	834	835
836	837	838	839	840
841	842	843	844	845
846	847	848	849	850
851	852	853	854	855
856	857	858	859	860
861	862	863	864	865
866	867	868	869	870
871	872	873	874	875
876	877	878	879	880
881	882	883	884	885
886	887	888	889	890
891	892	893	894	895
896	897	898	899	900
901	902	903	904	905
906	907	908	909	910
911	912	913	914	915
916	917	918	919	920
921	922	923	924	925
926	927	928	929	930
931	932	933	934	935
936	937	938	939	940
941	942	943	944	945
946	947	948	949	950
951	952	953	954	955
956	957	958	959	960
961	962	963	964	965
966	967	968	969	970
971	972	973	974	975
976	977	978	979	980
981	982	983	984	985
986	987	988	989	990
991	992	993	994	995
996	997	998	999	1000

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsserver.h	DATE 5/23/89	PAGE # 5/42
LINE #		HEADER TEXT		
481		extern void (*function_array[MAX_FUNCTION])();		
482		extern u_char fatal_hardware_error_encountered;		
483		extern u_char fatal_configuration_error_encountered;		
484		extern u_char non_fatal_configuration_error_encountered;		
485		extern long time_robot_was_lasped;		
486		extern long robot_delay_time;		
487		extern u_char robotting;		
488		extern u_char robottime;		
489		extern u_char robot_flag;		
490		typedef struct {		
491		u_long par_present;		
492		PAM_INFO *pam(MAX_PAM_COUNT);		
493		PEL_INFO *pel(MAX_PEL_COUNT);		
494		} LANE_INFO;		
495		/* Each lane has (MAX_PAM_COUNT) PAM_FB_BLOCK_COUNT structures */		
496		typedef struct {		
497		u_long pam_ptr[MAX_PAM_COUNT]; /* 1 ptrs past the last ptrs */		
498		u_long pam_fb_block_count[MAX_PAM_COUNT]; /* byte address */		
499		} PAM_FB_BLOCK_COUNT;		
500		PAM_FB_BLOCK_COUNT fb_block_count_array[MAX_LANE_COUNT];		
501		typedef struct {		
502		u_long tot_system_mem;		
503		u_long cpu_board_id;		
504		u_long clob_board_id;		
505		LANE_INFO *lane(MAX_LANE_COUNT);		
506		u_char slot_count; /* number of slots on 1 lane */		
507		u_char phy_lane_count; /* total physical lane in the system */		
508		} SYSTEM_INFO;		
509		typedef struct {		
510		PTEN_BITS *select_pal;		
511		PTEN_BITS *format0;		
512		PTEN_BITS *format1;		
513		PTEN_BITS *toggle;		
514		PTEN_BITS *adlamb;		
515		PTEN_BITS *adlamb1;		
516		PTEN_BITS *adlamb2;		
517		PTEN_BITS *adlamb3;		
518		PTEN_BITS *adlamb4;		
519		PTEN_BITS *adlamb5;		
520		PTEN_BITS *adlamb6;		
521		PTEN_BITS *adlamb7;		
522		PTEN_BITS *adlamb8;		
523		PTEN_BITS *adlamb9;		
524		PTEN_BITS *adlamb10;		
525		PTEN_BITS *adlamb11;		
526		PTEN_BITS *adlamb12;		
527		PTEN_BITS *adlamb13;		
528		PTEN_BITS *adlamb14;		
529		PTEN_BITS *adlamb15;		
530		PTEN_BITS *load_adm; /* used to load the ADEN/FORMAT registers */		
531		PTEN_BITS *ctl_out;		
532		PTEN_BITS *reset0;		
533		PTEN_BITS *reset1;		
534		PTEN_BITS *set0;		
535		PTEN_BITS *set1;		
536		PTEN_BITS *set2;		
537		PTEN_BITS *set3;		
538		PTEN_BITS *set4;		
539		PTEN_BITS *set5;		
540		PTEN_BITS *set6;		
541		PTEN_BITS *set7;		
542		PTEN_BITS *set8;		
543		PTEN_BITS *set9;		
544		PTEN_BITS *set10;		
545		PTEN_BITS *set11;		
546		PTEN_BITS *set12;		
547		PTEN_BITS *set13;		
548		PTEN_BITS *set14;		
549		PTEN_BITS *set15;		
550		PTEN_BITS *set16;		
551		PTEN_BITS *set17;		
552		PTEN_BITS *set18;		
553		PTEN_BITS *set19;		
554		PTEN_BITS *set20;		
555		PTEN_BITS *set21;		
556		PTEN_BITS *set22;		
557		PTEN_BITS *set23;		
558		PTEN_BITS *set24;		
559		PTEN_BITS *set25;		
560		PTEN_BITS *set26;		
561		PTEN_BITS *set27;		
562		PTEN_BITS *set28;		
563		PTEN_BITS *set29;		
564		PTEN_BITS *set30;		
565		PTEN_BITS *set31;		
566		PTEN_BITS *set32;		
567		PTEN_BITS *set33;		
568		PTEN_BITS *set34;		
569		PTEN_BITS *set35;		
570		PTEN_BITS *set36;		
571		PTEN_BITS *set37;		
572		PTEN_BITS *set38;		
573		PTEN_BITS *set39;		
574		PTEN_BITS *set40;		
575		PTEN_BITS *set41;		
576		PTEN_BITS *set42;		
577		PTEN_BITS *set43;		
578		PTEN_BITS *set44;		
579		PTEN_BITS *set45;		
580		PTEN_BITS *set46;		
581		PTEN_BITS *set47;		
582		PTEN_BITS *set48;		
583		PTEN_BITS *set49;		
584		PTEN_BITS *set50;		
585		PTEN_BITS *set51;		
586		PTEN_BITS *set52;		
587		PTEN_BITS *set53;		
588		PTEN_BITS *set54;		
589		PTEN_BITS *set55;		
590		PTEN_BITS *set56;		
591		PTEN_BITS *set57;		
592		PTEN_BITS *set58;		
593		PTEN_BITS *set59;		
594		PTEN_BITS *set60;		
595		PTEN_BITS *set61;		
596		PTEN_BITS *set62;		
597		PTEN_BITS *set63;		
598		PTEN_BITS *set64;		
599		PTEN_BITS *set65;		
600		PTEN_BITS *set66;		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lmsrver.h	DATE 5/23/89 TIME 1:20:20 pm	PAGE # 6/43
LINE #	HEADER TEXT			
601	#define even(x) ((x) & 1)			
602				
603				
604	/* The following is used if wordao and bitao are SHORT offset, i.e. if			
605	using PTH_BITS.			
606	#define CALC_PIN_NUMBER_SHORT(unitao, wordao, bitao) (unitao * 80 + \			
607	(79 - 16 * wordao) + \			
608	(bitao - 15))			
609				
610				
611	/* The following is used if wordao and bitao are LONG offset, i.e. if			
612	using PTH_BITS_LONGWORD.			
613	#define CALC_PIN_NUMBER_LONG(unitao, wordao, bitao) (unitao * 80 + \			
614	(79 - 32 * wordao) + \			
615	(bitao - 31))			
616				
617				
618	#define io_pin_transition(old_value, new_value, uninitialized_pin) \			
619	((new_value & (LOGIC_0 LOGIC_50)) & (old_value & LOGIC_21) ? \			
620	FALL_TRANSITION : \			
621	(new_value & (LOGIC_1 LOGIC_51)) & (old_value & LOGIC_20) ? \			
622	RISE_TRANSITION : \			
623	((uninitialized_pin == FALSE) ? NO_TRANSITION : \			
624	((new_value & (LOGIC_0 LOGIC_50)) ? FALL_TRANSITION : \			
625	((new_value & (LOGIC_1 LOGIC_51)) ? RISE_TRANSITION : NO_TRANSITION)))			
626				
627	#define input_pin_transition(old_value, new_value, uninitialized_pin) \			
628	((new_value & (LOGIC_0 LOGIC_50)) & \			
629	(old_value & (LOGIC_1 LOGIC_51 LOGIC_21)) ? \			
630	FALL_TRANSITION : \			
631	(new_value & (LOGIC_1 LOGIC_51)) & \			
632	(old_value & (LOGIC_0 LOGIC_50 LOGIC_20)) ? \			
633	RISE_TRANSITION : \			
634	((uninitialized_pin == FALSE) ? NO_TRANSITION : \			
635	((new_value & (LOGIC_0 LOGIC_50)) ? FALL_TRANSITION : \			
636	((new_value & (LOGIC_1 LOGIC_51)) ? RISE_TRANSITION : NO_TRANSITION)))			
637				
638				
639	typedef struct {			
640	u_char steady_state_fall_value;			
641	u_char two_state_value;			
642	u_char resolution;			
643	u_char expected_time;			
644	u_char set_by_user;			
645	} TM_RESULT;			
646				
647	extern TM_RESULT tm_result_array[];			
648	extern u_char			
649	is_measure_delay;			
650				
651	#define BOGUS_DEFINITION(user, def_ptr) \			
652	(((def_ptr) == NULL) \			
653	(((u_long)(def_ptr) >= (u_long)(user->definition[0]) & \			
654	((u_long)(def_ptr) < (u_long)(user->definition[(user->def_table_size)])))			
655				
656	/* inst_ptr is either			
657	* 1. NULL			
658	* 2. It points to somewhere in the table --> signify true instance.			
659				
660	#define BOGUS_INSTANCE(user, inst_ptr) \			
661	(((inst_ptr) == NULL) \			
662	(((u_long)(inst_ptr) >= (u_long)(user->instance[0]) & \			
663	((u_long)(inst_ptr) < (u_long)(user->instance[(user->inst_table_size)])))			
664				
665				
666	#define PTH_COUNT_ON_BLOCK(ptra_addr) \			
667	(((ptra_addr) - ((ptra_addr) & BLOCK_START_MASK)) / PTH_ADDR_INC + 1)			
668				
669	extern u_char read_loc_char();			
670	extern u_short read_loc_short();			
671	extern u_long read_loc_long();			
672	extern void write_loc_long();			
673				
674	#define REC_VALUE(low_limit, high_limit, value) \			
675	(((value) - (low_limit)) * 256 / ((high_limit) - (low_limit)))			
676				
677	#define fix_pel_sel(ptra, unitao, dab_ptr) \			
678	set_pel_sel(((PTH_BITS *) (ptra))>>ctl,			
679	(dab_ptr->unit_location[(unitao)].slot_no),			
680				
681	#define fix_pel_br(ptra) \			
682	set_pel_br(((PTH_BITS *) (ptra))>>ctl, BRANCH_NEVER),			
683				
684	#define fix_pel_ctl(ptra, unitao, dab_ptr) \			
685	{ u_short pel_ctl,			
686	pel_ctl = (((PTH_BITS *) (ptra))>>ctl & PEL_CTL_MASK) >> PEL_CTL_SHIFT;			
687	if ((pel_ctl == PEL_CTL_LOAD_DATA)			
688	(pel_ctl == PEL_CTL_LOAD_DATA_LAST_UNIT)) {			
689	if ((dab_ptr->unit_location[(unitao)].last_is_lane)			
690	set_pel_ctl(((PTH_BITS *) (ptra))>>ctl,			
691	PEL_CTL_LOAD_DATA_LAST_UNIT);			
692	else			
693	set_pel_ctl(((PTH_BITS *) (ptra))>>ctl,			
694	PEL_CTL_LOAD_DATA);			
695	}			
696				
697				
698	#define fix_pel_ctl_word(ptra, unitao, dab_ptr) \			
699	fix_pel_sel(ptra, (unitao), (dab_ptr))			
700	fix_pel_br(ptra)			
701	fix_pel_ctl(ptra, (unitao), (dab_ptr))			
702				
703	#define ADJUST_DELAY(val, time_scale, half_time_scale, divide_delay) \			
704	((val) - ((divide_delay) == FALSE ? (val) * (time_scale) :			
705	((val) + (half_time_scale)) / (time_scale)))			
706				
707	#define UNADJUST_DELAY(val, time_scale, half_time_scale, divide_delay) \			
708	((val) - (val) == 0 ? 0 : ((divide_delay) == FALSE ?			
709	(val) / (time_scale) :			
710	((val) * (time_scale) - (half_time_scale))))			
711				
712	extern u_char lm_printit;			
713				
714	#ifdef DEBUG			
715	#define DPRINTF(x) if (lm_printit == TRUE) (void)printf x			
716	else			
717	#define DPRINTF(x) /* do nothing */			
718	#endif			
719				
720	#define MAP_IN_LOGIC_VALUE(pin_number, has_resistor, value)			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/lmsrver.h	TIME	7/44
LINE #	HEADER TEXT			
721	if ((has_resistor) != NO_PULLUP_OR_PULLDOWN) {			
722	if ((value) & (LOGIC_20 LOGIC_21)) {			
723	if ((has_resistor) == PULLUP) {			
724	DPRINTF(("I conv 1->21 for PW: %d\n", (pin_number)));			
725	(value) = LOGIC_21;			
726	}			
727	else {			
728	DPRINTF(("I conv 1->20 for PW: %d\n", (pin_number)));			
729	(value) = LOGIC_20;			
730	}			
731	}			
732	}			
733	#define MAP_OUT_LOGIC_VALUE(pin_number, has_resistor, direction, value)			
734	if ((has_resistor) != NO_PULLUP_OR_PULLDOWN) {			
735	if ((value) & (LOGIC_20 LOGIC_21)) && (has_resistor == PULLUP) {			
736	DPRINTF(("O conv 20/1->21 for PW: %d\n", (pin_number)));			
737	(value) = LOGIC_21;			
738	}			
739	else if ((value) & (LOGIC_20 LOGIC_21)) &&			
740	(has_resistor == PULLDOWN) {			
741	DPRINTF(("O conv 20/1->20 for PW: %d\n", (pin_number)));			
742	(value) = LOGIC_20;			
743	}			
744	}			
745	}			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lnetwork.h	DATE 5/23/89 TIME 1:20:21 pm	PAGE # 1/45
LINE #	HEADER TEXT			
1	/* SCCS ID: lnetwork.h rev.3.1, 4/24/89 at 07:33:57 */			
2	#define RETSUCCESS -1			
3	#define RETERROR -2			
4	#define RETACCEPT -3			
5	/* We want to limit the total size of the network buffer to 1000 bytes during			
6	save and restore. The number of bytes to store pattern is 1000 minus the			
7	storage required for the header (see save_ptr_ase).			
8	*/			
9	#define SAVE_REST_PTR_BUFFER_LIMIT (1000 - 5 * sizeof(long))			
10	#define lm_put_int(val) (LM_CHK_PUT_LONG(lm_global_conn_ptr, (val)))			
11	#define lm_put_short(val) (LM_CHK_PUT_SHORT(lm_global_conn_ptr, (val)))			
12	#define lm_put_char(c) (LM_CHK_PUT_CHAR(lm_global_conn_ptr, (c)))			
13	#define lm_get_int() (LM_CHK_GET_LONG(lm_global_conn_ptr))			
14	#define lm_get_short() (LM_CHK_GET_SHORT(lm_global_conn_ptr))			
15	#define lm_get_char() (LM_CHK_GET_CHAR(lm_global_conn_ptr))			
16	/*			
17	#define lm_put_int(val) {\			
18	printf("Puti %08x --> %08x\n",\			
19	lm_global_conn_ptr->outgoing_buffer_pointer, (val));\			
20	LM_CHK_PUT_LONG(lm_global_conn_ptr, (val));\			
21	#define lm_put_short(val) {\			
22	printf("PutS %04x --> %04x\n",\			
23	lm_global_conn_ptr->outgoing_buffer_pointer, (val));\			
24	LM_CHK_PUT_SHORT(lm_global_conn_ptr, (val));\			
25	#define lm_put_char(c) {\			
26	printf("PutC %02x --> %02x\n",\			
27	lm_global_conn_ptr->outgoing_buffer_pointer, (c));\			
28	LM_CHK_PUT_CHAR(lm_global_conn_ptr, (c));\			
29	#define lm_get_int() {\			
30	printf("GetI %08x --> %08x\n",\			
31	lm_global_conn_ptr->incoming_buffer_pointer,\			
32	*((long *) (lm_global_conn_ptr->incoming_buffer_pointer));\			
33	LM_CHK_GET_LONG(lm_global_conn_ptr);\			
34	#define lm_get_short() {\			
35	printf("GetS %04x --> %04x\n",\			
36	lm_global_conn_ptr->incoming_buffer_pointer,\			
37	*((short *) (lm_global_conn_ptr->incoming_buffer_pointer));\			
38	LM_CHK_GET_SHORT(lm_global_conn_ptr);\			
39	#define lm_get_char() {\			
40	printf("GetC %02x --> %02x\n",\			
41	lm_global_conn_ptr->incoming_buffer_pointer,\			
42	*((char *) (lm_global_conn_ptr->incoming_buffer_pointer));\			
43	LM_CHK_GET_CHAR(lm_global_conn_ptr);\			
44	/*			
45	#define lm_get_out_chan_addr() (LM_CHK_ADDR(lm_global_conn_ptr))			
46	#endif DEBUG			
47	#define end_put(akt) {if (lm_printit == TRUE) printf("end_put, user: %d\n", lm_global_conn_ptr->id); lm_send_reply(lm_global_conn_ptr			
48);}			
49	#else			
50	#define end_put(akt) lm_send_reply(lm_global_conn_ptr)			
51	#endif			
52	#define reset_chan()			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lfsi.h	DATE 5/23/89	PAGE # 1/46
LINE #		HEADER TEXT		
1		/* SOCS ID: lfsi.h rev 3.1, 4/24/88 at 07:34:01 */		
2		#ifndef DESNG		
3		#define DREALLOC lm_realloc		
4		#define DFREE lm_free		
5		#define DALLOC lm_malloc		
6		#define DREALLOC lm_realloc		
7		#else		
8		#define DREALLOC realloc		
9		#define DFREE free		
10		#define DALLOC calloc		
11		#define DREALLOC malloc		
12		#endif		
13		#define LAST_COMPATIBLE_SPT_VERSION 0x01000000		
14		#define MAX_USER_COUNT 32		
15		#define MAX_PART_NAME_CHAR 20		
16		#define MAX_RETURN_STRING_LEN 256		
17		#define MAX_MODELER_NAME_CHAR 256		
18		#define MAX_DESNG 30		
19		#define MAXINT (((unsigned)0) >> 1)		
20		#define HOSTFILENAME "modelers.lfs"		
21		#define INVALID_KEY LM_INPUT+7		
22		#define MAX_PIN_TYPE 7		
23		#define MAX_LANE_COUNT 4		
24		#define MAX_PAN_COUNT 4		
25		#define MAX_SLOT_COUNT 8		
26		#define MAX_PIN_COUNT (MAX_LANE_COUNT * MAX_SLOT_COUNT * 80)		
27		#define TYPE_BIT_SHIFT 4		
28		#define MAX_PIN_NAME_LENGTH 100		
29		#define INVALID_PIN_NUMBER 0xffff		
30		#define DAS_STAT_PUBLIC 0		
31		#define DAS_STAT_PRIVATE 1		
32		#define MAX_PIN_ID 0x00000000		
33		#define PIN_TYPE_BIT_OFFSET 29		
34		#define MAX_LOGIC_STATE (LM_LOGIC_UNKNOWN - LM_LOGIC_ZERO + 2)		
35		#define MAX_OUTPUT_LOGIC_SIZE 128		
36		#define GET_PIN_VALUE_1 0		
37		#define GET_PIN_VALUE_2 1		
38		#define SAVE_SPT_VERSION_ID 4		
39		#define FILE_INTEGRITY_MARK 0xfabce07		
40		#define MAX_STRING_LENGTH 256		
41		#define MAX_SYSTEM_TABLE_SIZE 128		
42		#define MAX_NAME_LENGTH 32		
43		#define PTEN_COUNT_FUDGE_FACTOR 512		
44		#define NON_SPECIFIED_DELAY_VALUE 1 /* tick */		
45		/* pin type used in DAS_PIN */		
46		#define NONE 0		
47		#define IN 1		
48		#define OUT 2		
49		#define IO 3		
50		#define POWER 4		
51		#define GROUND 5		
52		#define NC 6		
53		/* pin class used in DAS_PIN */		
54		#define DATA 0		
55		#define EVAL 1		
56		#define STORE 2		
57		#define EDGE_RISE 3		
58		#define EDGE_FALL 4		
59		/* Possible values for pin values passed between host SPT and modeler */		
60		#define LOGIC_0 0x1		
61		#define LOGIC_1 0x2		
62		#define LOGIC_10 0x4		
63		#define LOGIC_11 0x8		
64		#define LOGIC_0 0x10		
65		#define LOGIC_10 0x20		
66		#define LOGIC_11 0x40		
67		#define LOGIC_1X 0x80		
68		#define LOGIC_1X 0xc0		
69		#define LOGIC_MAX_STATE LOGIC_11		
70		#define MAX_LOOP_NETWORK_TIMEOUT 600		
71		#define MAX_USER_PID_NAME_TABLE_SIZE 337 /* 19 */		
72		#define MAX_UNSPICED_DELAY_TABLE_SIZE 256		
73		#define NOT_OPENED 0		
74		#define UNAVAIL 1		
75		#define USED 2		
76		#define USED_FOR_RD_WR 3		
77		/* Constants used for test vector generation */		
78		#define NO_CMC 0		
79		#define DATA_CMC 1		
80		#define EVAL_OR_STORE_CMC 2		
81		#define STIMULUS_VECTOR 0		
82		#define RESULT_VECTOR 1		
83		#endif		
84		typedef struct {		
85		char *name;		
86		short fd; /* -1 if this host is not available */		
87		u_short state; /* NOT_OPENED, UNAVAIL, USED, USED_FOR_RD_WR */		
88		} SYSTEM_INFO;		
89		#endif		
90		typedef struct {		
91		char *name; /* name of the pin */		
92		char *pin_number_str; /* pin number string of the pin */		
93		char *pin_alias; /* key used to search -- */		
94		u_long key; /* consist of: 3-bit type and 29-bits pin-id */		
95		u_short pin_number; /* used in test vector generation */		
96		}		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lfsi.h	DATE 5/23/89	PAGE # 2/47
LINE #	HEADER TEXT			
121	/* unsigned : 3; /* used in sorting for test vector generation */			
122	/* unsigned processed: 1; /* DATA, EVAL, STORE, EDGE_RISE, EDGE_FALL */			
123	/* unsigned pin_class: 3; /* MON, TN, OUT, IO, POWER, CROWD, NC */			
124	/* unsigned pin_type: 3; /* MON, TN, OUT, IO, POWER, CROWD, NC */			
125	/* u_short line_index; /* used in test vector generation */			
126	/* DAB_PID; /*			
127	/* If you change the following structure be sure to change the in_insert_key			
128	/* procedure: */			
129	/*			
130	typedef struct user_pid {			
131	struct user_pid *next_in_bucket;			
132	u_long key;			
133	u_long dab_pid_table_index;			
134	} USER_PID;			
135	/*			
136	typedef struct {			
137	u_long minimum;			
138	u_long typical;			
139	u_long maximum;			
140	} MIN_TYP_MAX;			
141	/*			
142	typedef struct {			
143	u_char in_value; /* current value of the pin */			
144	u_char out_value; /* current value of output pin */			
145	u_long min_delay; /* MIN/TYP delays of the pin */			
146	u_long typ_delay;			
147	u_long max_delay;			
148	u_long sim_time; /* simulation time for this INPUT pin change */			
149	short input_change; /* links all of the input pin changes */			
150	short output_change; /* links all of the output pin changes */			
151	/*			
152	unsigned just_went_float: 1; /* indicates that the pin just went to 1 */			
153	/* indicates that the pin is in input_chg_list */			
154	unsigned is_linked: 1; /* IN_TABLE DELAY, IN_MEASURED DELAY */			
155	u_char delay_type; /* IN_COMPOSITE DELAY, IN_UNSPECIFIED DELAY */			
156	/*			
157	u_short event_pin_number; /* pin number that causes the output to change */			
158	} PIN_INFO;			
159	/*			
160	typedef struct instance_info {			
161	FILE *vector_file;			
162	FILE *ts_file;			
163	char *vector_buffer;			
164	struct instance_info *link; /* used to free instance info structures */			
165	/* also used to link all instances for the */			
166	/* save definition during save operation */			
167	struct instance_info *fault_link; /* used to link all faults derived from */			
168	/* the same instance during save operation */			
169	/*			
170	PIN_INFO **pin_table;			
171	long inst_pin_change_time; /* used in test vector generation */			
172	time of first evaluation;			
173	char *device_info_string; /* physical device name of the instance */			
174	box_id; /* File descriptor of the system */			
175	short inst_id; /* inst/var id which is used by the user */			
176	short assoc_inst_id; /* used during save operation to find the */			
177	/* instance gives a fault */			
178	short box_inst_id; /* inst/var id which is used in the module */			
179	short assoc_box_inst_id; /* 0 means that this is the box instance id */			
180	/* which this fault is derived from */			
181	short def_id; /* index to the definition id table */			
182	u_short data_pin_change_count;			
183	u_short eval_pin_change_count;			
184	u_short store_pin_change_count;			
185	short data_pin_change_list;			
186	short eval_pin_change_list;			
187	short store_pin_change_list;			
188	short output_change_list;			
189	u_char some_eval_or_store_pin;			
190	/* NO_CBG, DATA_CBG, EVAL_OR_STORE_CBG */			
191	type_of_pin_change;			
192	short pin_change_count; /* For vector logging */			
193	u_char get_pin_change_called;			
194	} INSTANCE_INFO;			
195	/* Structure for unspecified delays */			
196	typedef union {			
197	struct {			
198	unsigned from_state:4;			
199	unsigned from_pin:12;			
200	unsigned to_state:4;			
201	unsigned to_pin:12;			
202	} field;			
203	u_long unspecified_value;			
204	u_long unspecified_delay_type;			
205	/* Unspecified specified delay linked list structure */			
206	typedef struct unspecified_delay_list_element {			
207	unspecified_delay_type unspecified_delay;			
208	struct unspecified_delay_list_element *next;			
209	} unspecified_delay_list_element;			
210	/*			
211	typedef struct {			
212	char *device_name;			
213	SYSTEM_INFO *system_ptr;			
214	DAB_PID *dab_pid_table;			
215	USER_PID *user_pid_table[MAX_USER_PID_HASH_TABLE_SIZE];			
216	INSTANCE_INFO *instance_link; /* used during save operation to point to */			
217	/* the head of the instance linked-list */			
218	MIN_TYP_MAX *delay_table;			
219	short input_pin_number_returned;			
220	short output_pin_number_returned;			
221	short io_pin_number_returned;			
222	short power_pin_number_returned;			
223	short ground_pin_number_returned;			
224	short sc_pin_number_returned;			
225	u_short have_table_element_count;			
226	u_short dab_pid_table_size;			
227	u_short def_id;			
228	u_short box_def_id;			
229	u_short lowest_pin_name;			
230	u_short real_pin_count;			
231	u_short lowest_eval_or_store_pinno; /* used to return time 0 val */			
232	u_char table_validated;			
233	u_char use_default;			
234	u_char report_size;			
235	unspecified_delay_list_element *unspecified_delay_list;			
236	} DEFINITION_INFO;			
237	/*			
238	typedef struct {			
239	u_long word[3];			
240	} PTRS_BITS_LONGWORD;			

Copyright 1989 Logic Modeling Systems	HEADER FILE include/lmfi.h	DATE 5/23/89 TIME 1:20:21 pm	PAGE # 3/48
LINE #	HEADER TEXT		
241	/*-----*/		
242	typedef struct {		
243	long user_value;		
244	u_short lm_value;		
245	u_short net;		
246	} USER_TO_LM_LOGIC_LEVEL;		
247	/*-----*/		
248	typedef struct file_info {		
249	struct file_info *next;		
250	char *name;		
251	FILE *file_ptr;		
252	u_short instance_id;		
253	} FILE_INFO;		
254	/*-----*/		
255	#define MAX_CONN_BUFFER_SIZE 1024		
256	#define MAX_CONN_BUFFER_SIZE 1024		
257	/*-----*/		
258	extern CONNECTION *lm_table_of_conns();		
259	extern CONNECTION *lm_global_conn_ptr;		
260	extern char lm_global_conn_ptr;		
261	extern char lm_global_conn_ptr;		
262	#define lm_put_int(val) LM_CHK_PUT_LONG(lm_global_conn_ptr, (int)(val))		
263	#define lm_put_short(val) LM_CHK_PUT_SHORT(lm_global_conn_ptr, (short)(val))		
264	#define lm_put_char(c) LM_CHK_PUT_CHAR(lm_global_conn_ptr, (char)(c))		
265	#define lm_get_int() LM_CHK_GET_LONG(lm_global_conn_ptr)		
266	#define lm_get_short() LM_CHK_GET_SHORT(lm_global_conn_ptr)		
267	#define lm_get_char() LM_CHK_GET_CHAR(lm_global_conn_ptr)		
268	/* Network Debouncing macros */		
269	#define lm_put_int(val) {		
270	printf("Put %d\n", val);		
271	lm_global_conn_ptr->outgoing_buffer_pointer = (val);		
272	LM_CHK_PUT_LONG(lm_global_conn_ptr, (val));		
273	#define lm_put_short(val) {		
274	printf("Put %d\n", val);		
275	lm_global_conn_ptr->outgoing_buffer_pointer = (val);		
276	LM_CHK_PUT_SHORT(lm_global_conn_ptr, (val));		
277	#define lm_put_char(c) {		
278	printf("Put %c\n", c);		
279	lm_global_conn_ptr->outgoing_buffer_pointer = (c);		
280	LM_CHK_PUT_CHAR(lm_global_conn_ptr, (c));		
281	#define lm_get_int() {		
282	printf("Get %d\n", val);		
283	lm_global_conn_ptr->incoming_buffer_pointer =		
284	*((long *) (lm_global_conn_ptr->incoming_buffer_pointer));		
285	LM_CHK_GET_LONG(lm_global_conn_ptr);		
286	#define lm_get_short() {		
287	printf("Get %d\n", val);		
288	lm_global_conn_ptr->incoming_buffer_pointer =		
289	*((short *) (lm_global_conn_ptr->incoming_buffer_pointer));		
290	LM_CHK_GET_SHORT(lm_global_conn_ptr);		
291	#define lm_get_char() {		
292	printf("Get %c\n", c);		
293	lm_global_conn_ptr->incoming_buffer_pointer =		
294	*((char *) (lm_global_conn_ptr->incoming_buffer_pointer));		
295	LM_CHK_GET_CHAR(lm_global_conn_ptr);		
296	/*		
297	#define lm_choose_conn(akt) {		
298	if ((akt) == -1) (lm_table_of_conns((akt)) == NULL) {		
299	lm_queue_message(ERROR_MSG, "Invalid socket number: %d", akt);		
300	return(LM_ERROR);		
301	lm_global_conn_ptr = lm_table_of_conns((akt));		
302	#define lm_end_put(akt) lm_send_request_get_reply(lm_global_conn_ptr)		
303	/*-----*/		
304	/* Simulation tables */		
305	extern SYSTEM_INFO *lm_system_table;		
306	extern INSTANCE_INFO *lm_inst_id_table;		
307	extern DEFINITION_INFO *lm_def_id_table;		
308	extern INSTANCE_INFO *lm_inst_id_table;		
309	extern u_short lm_inst_id_table_size;		
310	extern u_short lm_inst_id_table_size;		
311	extern short lm_def_id_avail;		
312	/* Pointers to Entities selected */		
313	extern SYSTEM_INFO *lm_system_selected;		
314	extern DEFINITION_INFO *lm_definition_selected;		
315	extern INSTANCE_INFO *lm_instance_selected;		
316	extern u_long lm_inst_id_specified;		
317	extern u_long lm_def_id_specified;		
318	/* Logic level mapping */		
319	extern USER_TO_LM_LOGIC_LEVEL lm_user_to_lm_logic_level_map[];		
320	extern u_short lm_to_lm_logic_level_map[];		
321	extern u_short lm_logic_level_map_count;		
322	/* Temporary buffer for DART processing */		
323	extern char lm_temp_buffer;		
324	extern char lm_thef_ptr;		
325	extern char lm_max_thef_addr;		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/lsm.h	DATE 5/23/89 TIME 1:20:21 pm	PAGE # 4/49
LINE #	HEADER TEXT			
361	/* Simulation characteristics */			
362	extern u_short lm_sim_verified,			
363	extern u_long lm_simulation_time,			
364	extern u_long lm_gol_simulator_type,			
365	extern u_long lm_gol_time_scale_number,			
366	extern u_long lm_gol_time_scale_units,			
367	extern u_char lm_gol_divide_delay,			
368	extern u_long lm_gol_time_scale,			
369	extern u_long lm_gol_half_time_scale,			
370	extern u_short lm_host_file_opened,			
371	extern FILE *tm_file,			
372	#ifdef DEBUG			
373	extern u_char lm_debug[],			
374	#endif			
375	extern u_long lm_network_timeout,			
376	extern FILE_INFO *lm_vector_file_ptr,			
377	extern FILE_INFO *lm_timing_file_ptr,			
378	/*			
379	*/			
380	#define NOCUS_INSTANCE(inst_ptr) \			
381	(((inst_ptr) == NULL) \			
382	(((u_long)(inst_ptr) >= (u_long)lm_inst_id_table[0]) && \			
383	(((u_long)(inst_ptr) < (u_long)lm_inst_id_table[lm_inst_id_table_size])))			
384	#endif			
385	#define DPRINTF(x) (void)printf x			
386	#else			
387	#define DPRINTF(x) /* do nothing */			
388	#endif			
389	#define clear_errors lm_flush_message_queue			
390	#define XISDIGIT(X) (((X) >= '0') && ((X) <= '9')) ? 1 : 0			
391	#define ADJUST_DELAY(val, time_scale, half_time_scale, divide_delay) \			
392	((val) - ((divide_delay) == FALSE ? (val) * (time_scale) : \			
393	((val) + (half_time_scale)) / (time_scale)))			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/magic.h	5/23/89	1/50
		TIME	1:20:21 pm	

LINE #	HEADER TEXT
1	/* SCS3 ID: magic.h rev 1.1, 4/24/89 at 07:34:05 */
2	
3	#define EMPTY_192_BYTES unsigned \
4	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
5	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
6	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
7	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
8	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
9	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32, \
10	/*
11	~ All magic chip registers are one bit per data channel, except
12	~ the status register (parity).*/
13	*/
14	typedef union {
15	struct
16	{
17	unsigned :16,
18	data_out:16,
19	/* formatted data out */
20	}
21	unsigned :16,
22	md_output_enable:16,
23	/* formatted medium drive output enable */
24	unsigned :16,
25	hd_output_enable:16,
26	/* formatted hard drive output enable */
27	unsigned :16,
28	control_output:16,
29	/* last bit for the control shift register */
30	unsigned :16,
31	last_cycle:16,
32	/* soft drive output enable */
33	unsigned :16,
34	hard_enable:16,
35	/* hard/medium drive enable */
36	unsigned :16,
37	data_format:16,
38	/* input data format bit */
39	unsigned :16,
40	error_data:16,
41	/* last data before ERROR is asserted */
42	unsigned :16,
43	short_sample:16,
44	/* short circuit detection */
45	unsigned :16,
46	unknown_sample:16,
47	/* sampled V values on out */
48	unsigned :16,
49	float_sample:16,
50	/* sampled IIS values on out */
51	unsigned :16,
52	logic_sample:16,
53	/* sampled logic values on out */
54	unsigned :16,
55	ttl_sample:16,
56	/* sampled TTL logic values on out */
57	unsigned :32,
58	/* unused */
59	/* status register */
60	unsigned :16,
61	:14,
62	/* unused
63	parity_out:1,
64	/* output parity
65	parity_in:1,
66	/* input parity
67	unsigned :16,
68	reset:16,
69	/* reading causes an internal magic reset */
70	EMPTY_192_BYTES
71	};
72	unsigned long reg[16];
73	/* MAGIC_CHIP,

COPYRIGHT 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/message.h	5/23/89	1/51
		TIME	1:20:21 pm	
LINE #	HEADER TEXT			
1	/* SCCS ID: message.h rev 3.1, 4/24/89 at 07:34:08 */			
2	/*			
3	The following #defines are the types of messages that in_queue_message() queues:			
4	/*			
5	SYS_ types are used to pick up the host's operating system's error			
6	string as defined by the operating system (VMS, UNIX, ...). This			
7	operating system error string is appended to the string that is created			
8	from the rest of in_queue_message()'s parameters.			
9	/*			
10	In UNIX, any function that sets "errno" after failure should be followed by a			
11	call to in_queue_message(SYS_ERROR_MSG, ...) or in_queue_message(SYS_WARNING_MSG, ...)			
12	depending upon your code's view of the system calls failure.			
13	/*			
14	VMS handles all of the C library functions in a UNIX compatible way, with			
15	special handling for VMS system services (that's the VMS_ MSGs defined below).			
16	See me if this concerns you as the calls to in_queue_message() are different			
17	for these message types.			
18	/*			
19	We will make VMS compatible with UNIX.			
20	/*			
21	By using the SYS_ types, the operating system dependent handling of error			
22	indicators is far removed from your application code and thereby localised			
23	and easily ported and/or changed.			
24	/*			
25	The ERROR_MSG and WARNING_MSG types are for messages that originate because			
26	of errors and warnings that are detected by your application code functionality.			
27	/*			
28	#define ERROR_MSG 1			
29	#define WARNING_MSG 2			
30	#define SYS_ERROR_MSG 3			
31	#define VMS_ERROR_MSG 4			
32	#define SYS_WARNING_MSG 5			
33	#define VMS_WARNING_MSG 6			
34	/*			
35	The size of the maximum message that in_queue_message() will ever			
36	format into its string argument. See the examples below for usage.			
37	/*			
38	#define MAX_MESSAGE 256			
39	/*			
40	Routine to queue messages:			
41	extern void in_queue_message();			
42	/*			
43	Routine to dequeue messages:			
44	extern u_short in_dequeue_message();			
45	/*			
46	Routine to clear out all pending messages.			
47	It's use should be very carefully considered.			
48	/*			
49	extern void in_flush_message_queue();			
50	/*			
51	Routine to return what message types are currently queued.			
52	/*			
53	extern void in_message_types();			
54	/*			
55	EXAMPLES:			
56	/*			
57	All these examples are taken from the network handling code. There are			
58	many other examples in their larger context that you should feel free			
59	to peruse as you feel necessary. See "/usr/include/netinet.h".			
60	/*			
61	1. in_queue_message(ERROR_MSG, "too many attempts(%d) to contact node(%d) giving up", MAX_RETRY_ATTEMPTS);			
62	/*			
63	2. in_queue_message(WARNING_MSG, "message transmission retry on time out(%d)", network_timeout);			
64	/*			
65	3. in_queue_message(SYS_ERROR_MSG, "failure to send %d bytes to the node(%d), only sent %d", byte_count, node);			
66	/*			
67	4. a short type:			
68	char str[MAX_MESSAGE];			
69	/*			
70	while(in_dequeue_message(&str) != FAILURE) {			
71	if (type == ERROR_MSG)			
72	(void) fprintf(stderr, "ERROR: %s", str);			
73	else (void) fprintf(stderr, "WARNING: %s", str);			
74	}			
75	/*			
76	NOTES:			
77	/*			
78	1. There are to be no "error" or "warning" string prefixes on any of the			
79	message strings that are passed to in_queue_message(). The "type" is			
80	used to indicate severity with the caller of in_queue_message().			
81	deciding what an appropriate and consistent message prefix is:			
82	/*			
83	2. There is to be no newline, "\n", character appended to any of the message			
84	strings that are passed to in_queue_message().			
85	/*			
86	3. The rest of the in_queue_message() parameters are anything that sprintf()			
87	will take - starting with the format string and ending with a list of the			
88	variables that supply values for the % format specifiers.			
89	/*			
90	4. The first parameter to in_queue_message() must be one of the first four			
91	#defines listed above.			
92	/*			
93	5. The distinction between ERROR_MSG and SYS_ERROR_MSG is a critical one and			
94	must be maintained in your code's usage of in_queue_message().			
95	/*			
96	/*			
97	/*			
98	/*			
99	/*			
100	/*			
101	/*			
102	/*			
103	/*			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/mod_def.h	DATE 5/23/89	PAGE # 1/52
LINE #		HEADER TEXT		
1		/* SOC8_ID: mod_def.h rev 1.1, 4/24/89 at 07:34.11 */		
2		/* Modular Definitions */		
3		/*		
4		* There are four lanes in the UP-1000: A, B, C, and D.		
5		* Each lane contains slots for eight FELs.		
6		* The first 64Kb address for each lane is:		
7		/*		
8		* 0x00000000 for Lane A		
9		* 0x00000000 for Lane B		
10		* 0x00000000 for Lane C		
11		* and 0x00000000 for Lane D		
12		* */		
13		/*		
14		*define NUMBER_OF_LANES 4		
15		*define NUMBER_OF_SLOTS 8		
16		*define NUMBER_OF_FELS NUMBER_OF_SLOTS		
17		*define LANE_SIZE (u_long)0x10000000 /* 256 Kbytes */		
18		*define LANE_A_OFFSET (u_long)0x00000000		
19		*define LANE_B_OFFSET (LANE_A_OFFSET + LANE_SIZE)		
20		*define LANE_C_OFFSET (LANE_B_OFFSET + LANE_SIZE)		
21		*define LANE_D_OFFSET (LANE_C_OFFSET + LANE_SIZE)		
22		/*		
23		*define FEL_ID_FROM 0x00000000		
24		*define LANE_A_FEL_ID_FROM 0x00100000		
25		*define LANE_B_FEL_ID_FROM 0x00200000		
26		*define LANE_C_FEL_ID_FROM 0x00300000		
27		*define LANE_D_FEL_ID_FROM 0x00400000		
28		/*		
29		*define SEND (u_long)0x00000000 /* Default random seed */		
30		*define DING_MAX_INPUT 256		
31		*define MONE 0		
32		*define TIMER_RES 5 /* Timer resolution = 5 ms */		
33		/* Modular Macros */		
34		/*		
35		*define Lane_code(X) (1 << (X))		
36		/*		
37		*define Clear_key_buf() while(!is_check_key() != 0) is_get_key()		
38		/*		
39		* Global declarations of functions that do not return integers */		
40		/*		
41		void pec_play_cleanup();		
42		void pec_configure_all_pacs();		
43		void pec_info_init();		
44		void pec_play_cleanup();		
45		void pec_set_first_block();		
46		void pec_probe_all_pacs();		
47		void diag_get_log();		
48		void diag_get_ulog();		
49		void diag_get_uhan();		
50		/*		
51		*/		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/mod_err.h	DATE 5/23/89	PAGE # 1/53
LINE #		HEADER TEXT		
1		/* SOCS_ID: mod_err.h rev 1.1, 4/24/89 at 07:34:14 */		
2		typedef struct {		
3		unsigned pel_error_list:32; /* WHICH PEL HAS ERROR */		
4		unsigned		
5		#ifdefined VAX		
6		error:1; /* Error detected */		
7		lanes_errors:4; /* Lanes A-D have errors */		
8		pec_lane_errors:4; /* pec lanes have errors */		
9		dab_change:1; /* dab inserted/removed */		
10		tmg_error:1; /* TMC error */		
11		unknown_source_of_interrupt:1; /* source of interrupt unknown */		
12		lanes_enable:4;		
13		lanes_a_pel_control:3;		
14		lanes_a_data_valid:1;		
15		lanes_c_pel_control:3;		
16		lanes_c_data_valid:1;		
17		lanes_d_pel_control:3;		
18		lanes_d_data_valid:1;		
19		lanes_e_pel_control:3;		
20		lanes_e_data_valid:1;		
21		lanes_f_pel_control:3;		
22		lanes_f_data_valid:1;		
23		#else		
24		lanes_c_data_valid:1;		
25		lanes_c_pel_control:3;		
26		lanes_d_data_valid:1;		
27		lanes_d_pel_control:3;		
28		lanes_e_data_valid:1;		
29		lanes_e_pel_control:3;		
30		lanes_f_data_valid:1;		
31		lanes_f_pel_control:3;		
32		lanes_enable:4;		
33		unknown_source_of_interrupt:1; /* source of interrupt unknown */		
34		tmg_error:1; /* TMC error */		
35		dab_change:1; /* dab inserted/removed */		
36		pec_lane_errors:4; /* pec lanes have errors */		
37		lanes_errors:4; /* Lanes A-D have errors */		
38		error:1; /* Error detected */		
39		#endif		
40		struct		
41		{		
42		#ifdefined VAX		
43		unsigned		
44		pec_refresh_error:1;		
45		pec_request_error:1;		
46		pec_patterns_error:1;		
47		pec_control_word_parity_error:1;		
48		pec_high_word_parity_error:1;		
49		pec_low_word_parity_error:1;		
50		pec_branch_address:15;		
51		pec_block_offset:8;		
52		ped:3;		
53		unsigned		
54		pec_parity_error_address:26;		
55		ped:6;		
56		#else		
57		unsigned		
58		ped:3;		
59		pec_block_offset:8;		
60		pec_branch_address:15;		
61		pec_low_word_parity_error:1;		
62		pec_high_word_parity_error:1;		
63		pec_control_word_parity_error:1;		
64		pec_patterns_error:1;		
65		pec_request_error:1;		
66		pec_refresh_error:1;		
67		unsigned		
68		ped:6;		
69		pec_parity_error_address:26;		
70		#endif		
71		} pec_error[4]; /* which pec has errors */		
72		struct		
73		{		
74		#ifdefined VAX		
75		unsigned		
76		dab_inserted:1;		
77		dab_removed:1;		
78		play_error:1;		
79		any_magic_short:1;		
80		any_magic_parity:1;		
81		magic_parity_out:5;		
82		ped5:21;		
83		#else		
84		unsigned		
85		ped5:21;		
86		magic_parity_out:5;		
87		any_magic_parity:1;		
88		any_magic_short:1;		
89		play_error:1;		
90		dab_removed:1;		
91		dab_inserted:1;		
92		#endif		
93		unsigned short magic_short[5];		
94		} pel_error[32]; /* errors per PEL */		
95		} LM_HARDWARE_ERROR;		
96		typedef struct {		
97		#ifdefined VAX		
98		unsigned do_tmg_calibration: 1;		
99		unsigned ped: 25;		
100		unsigned tmg_cal: 1;		
101		unsigned no_tmg:1;		
102		unsigned no_pem_for_pec:4;		
103		#else		
104		unsigned no_pem_for_pec:4;		
105		unsigned no_tmg:1;		
106		unsigned tmg_cal: 1;		
107		unsigned ped: 25;		
108		unsigned do_tmg_calibration: 1;		
109		#endif		
110		u_short pem_strapped_wrong;		
111		u_short pem_stacked_wrong;		
112		u_long duplicate_segment;		
113		u_long missing_segment;		
114		u_long no_pel_for_dab;		
115		u_long no_pem_for_dab;		
116		u_long illegal_duplicate_device;		
117		u_long device_too_large; /* more than MAX_UNIT_COUNT units */		
118		u_long illegal_pel_stacking;		
119		}		
120				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/mod_err.h	DATE 5/23/89	PAGE # 2/54
LINE #	HEADER TEXT			
21) CONFIGURATION_ERRORS,			
22				
23	typedef struct {			
24	u_long db_inaction_count; /* has to be the 1st one */			
25	u_long pattern_count_mal,			
26	u_long pattern_count_lal,			
27	u_long longest_pattern_seq,			
28	u_long definition_count,			
29	u_long instance_count,			
30	u_long fault_count,			
31) RUNTIME_STAT_STRUCT;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/network.h	DATE 5/23/89	PAGE # 1/55
LINE #		HEADER TEXT		
1		/* SOCS_ID: network.h rev 3.1.1, 4/24/89 at 07:34:13 */		
2		/* The size of socket_address must be equal that of sockaddr. */		
3		/*		
4		typedef struct socket_address {		
5		short family;		
6		short port;		
7		long address;		
8		u_char destination[6];		
9		u_short packet_type;		
10		} SOCKET_ADDRESS;		
11		/* MSG_PEEK is taken directly from UNIX and can not be changed from 0x3. */		
12		/*		
13		#define MSG_PEEK 0x3 /* peek at incoming message */		
14		/*		
15		/* Used for timeouts on the modular with recvfrom().		
16		/* This value must remain exclusive of MSG_PEEK. */		
17		/*		
18		#define RECVFROM_TIMEOUT 0x3		
19		/* Used as a flag in the modular's version of sendto(). */		
20		#define RAW_PACKET 0x4		
21		#define UDP_IP_PACKET 0x8		
22		#define ALIVE_PACKET 0x10		
23		#define DEATH_PACKET 0x20		
24		/* Are You Alive commands */		
25		#define AYA_INQUIRE 0x100		
26		#define AYA_KILL 0x102		
27		#define AYA_INTERRUPT 0x104		
28		#define SOCK_SIZE sizeof(SOCKET_ADDRESS)		
29		/*		
30		/* The following must exactly match their equivalents in		
31		/* the UNIX files <sys/socket.h>, <netinet/in.h>,		
32		/* <netdb.h> and <sys/time.h>. */		
33		/*		
34		#define AF_INET 2 /* internet: UDP, TCP, etc. */		
35		#define SOCK_DGRAM 2 /* datagram socket */		
36		/*		
37		#ifndef MODULE		
38		#define INADDR_ANY 0x00000000		
39		#endif		
40		/*		
41		/* Structure used by kernel to store host addresses. */		
42		/*		
43		#ifndef PC186		
44		struct sockaddr {		
45		short sa_family; /* address family */		
46		char sa_data[14]; /* up to 14 bytes of direct address */		
47		};		
48		#endif		
49		/*		
50		/* End of the required UNIX matching items. */		
51		/*		
52		/* An arbitrary port number that can be anything greater than 1024. */		
53		/*		
54		#define MODELS_ETHNET_PORT 2010		
55		#define MODELS_OUT_OF_BAND_PORT 2012		
56		#define MODELS_ARE_YOU_ALIVE_PORT 2014		
57		#define MODELS_DEATH_PORT 2016		
58		/*		
59		typedef struct ethernet_header {		
60		u_char destination[6];		
61		u_char source[6];		
62		u_short type;		
63		} ETHERNET_HEADER;		
64		/*		
65		typedef struct ip_header {		
66		u_char version_and_header_length;		
67		u_char type_of_service;		
68		u_short total_length;		
69		u_short identification;		
70		u_short fragment_offset;		
71		u_char time_to_live;		
72		u_char protocol;		
73		u_short checksum;		
74		u_long source_address;		
75		u_long destination_address;		
76		} IP_HEADER;		
77		/*		
78		typedef struct udp_header {		
79		u_short source_port;		
80		u_short destination_port;		
81		u_short length;		
82		u_short checksum;		
83		} UDP_HEADER;		
84		/*		
85		typedef struct lm_header {		
86		u_long byte_count;		
87		u_long process_id;		
88		u_long sequence_number;		
89		u_long partial_packet_count;		
90		} LM_HEADER;		
91		/*		
92		typedef struct packet_header {		
93		#ifndef MODULE		
94		ETHERNET_HEADER ethernet_hdr;		
95		IP_HEADER ip_hdr;		
96		UDP_HEADER udp_hdr;		
97		#endif		
98		LM_HEADER lm_hdr;		
99		} PACKET_HEADER;		
100		/*		
101		typedef struct connection {		
102		int fd;		
103		char *name;		
104		/*		
105		u_short on_sending;		
106		u_short on_receiving;		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/network.h	DATE 5/23/89	PAGE # 3/57
HEADER TEXT				
LINE #				
240	#define LM_PUT_CHAR AT MARK(M, X, Y)			
241	((u_char *)((u_long)((X)->outgoing_buffer_base) + (u_long)(M)) = (u_char)(Y))			
242	/* (u_char *)((u_long)((X)->incoming_buffer_pointer)			
243	#define LM_GET_ADDR(X) ((X)->incoming_buffer_pointer)			
244	#define LM_ADVANCE_ADDR(X, Y) ((X)->incoming_buffer_pointer) += (Y)			
245	/* Machine dependent get and put macros */			
246	/* Machine dependent get and put macros */			
247	/* Machine dependent get and put macros */			
248	#ifdef SUN4			
249	#define NET_BYTE_ALIGN			
250	#endif			
251	#ifdef APOLLO10K			
252	#define NET_BYTE_ALIGN			
253	#endif			
254	#ifdef VMS			
255	#define NET_BYTE_ALIGN			
256	#endif			
257	#ifdef NET_BYTE_ALIGN /* SUN4, APOLLO10K, VMS */			
258	#define LM_GET_SHORT(X) ((LM_GET_CHAR(X) < 8) \			
259	LM_GET_CHAR(X))			
260	/* (LM_GET_CHAR(X) < 8) \			
261	#define LM_PUT_SHORT(X, Y) (LM_PUT_CHAR(X, ((Y) >> 8)), \			
262	LM_PUT_CHAR(X, (Y)))			
263	/* (LM_PUT_CHAR(X, ((Y) >> 8)), \			
264	#define LM_PUT_SHORT_AT_MARK(M, X, Y) (LM_PUT_CHAR_AT_MARK(((M)+0), X, ((Y) >> 8)), \			
265	LM_PUT_CHAR_AT_MARK(((M)+1), X, (Y)))			
266	/* (LM_PUT_CHAR_AT_MARK(((M)+1), X, (Y)))			
267	#define LM_GET_LONG(X) ((LM_GET_CHAR(X) < 24) \			
268	(LM_GET_CHAR(X) < 16) \			
269	(LM_GET_CHAR(X) < 8) \			
270	LM_GET_CHAR(X))			
271	/* (LM_PUT_CHAR(X, ((Y) >> 24)), \			
272	#define LM_PUT_LONG(X, Y) (LM_PUT_CHAR(X, ((Y) >> 24)), \			
273	LM_PUT_CHAR(X, ((Y) >> 16)), \			
274	LM_PUT_CHAR(X, ((Y) >> 8)), \			
275	LM_PUT_CHAR(X, (Y)))			
276	/* (LM_PUT_CHAR(X, ((Y) >> 24)), \			
277	#define LM_PUT_LONG_AT_MARK(M, X, Y) (LM_PUT_CHAR_AT_MARK(((M)+0), X, ((Y) >> 24)), \			
278	LM_PUT_CHAR_AT_MARK(((M)+1), X, ((Y) >> 16)), \			
279	LM_PUT_CHAR_AT_MARK(((M)+2), X, ((Y) >> 8)), \			
280	LM_PUT_CHAR_AT_MARK(((M)+3), X, (Y)))			
281	/* (LM_PUT_CHAR_AT_MARK(((M)+3), X, (Y)))			
282	#else /* NET_BYTE_ALIGN */			
283	/* NET_BYTE_ALIGN */			
284	#ifdef PC386			
285	u_long stohl(), htonl();			
286	u_short stohs(), htons();			
287	/* (X)->incoming_buffer_pointer = sizeof(short); \			
288	#define LM_GET_SHORT(X) (stohs(*((u_short *)((X)->incoming_buffer_pointer - sizeof(short))))			
289	/* (X)->incoming_buffer_pointer = sizeof(long); \			
290	#define LM_GET_LONG(X) (stohl(*((u_long *)((X)->incoming_buffer_pointer - sizeof(long))))			
291	/* (X)->incoming_buffer_pointer = sizeof(long); \			
292	#define LM_PUT_SHORT(X, Y) ((*((u_short *)((X)->outgoing_buffer_pointer = htons((Y))), \			
293	(X)->outgoing_buffer_pointer += sizeof(short))			
294	/* (X)->outgoing_buffer_pointer = sizeof(long); \			
295	#define LM_PUT_LONG(X, Y) ((*((u_long *)((X)->outgoing_buffer_pointer = htonl((Y))), \			
296	(X)->outgoing_buffer_pointer += sizeof(long))			
297	/* (X)->outgoing_buffer_pointer = sizeof(long); \			
298	#define LM_PUT_SHORT_AT_MARK(M, X, Y) ((*((u_short *)((X)->outgoing_buffer_base + (M)) = htons((Y)))			
299	/* (X)->outgoing_buffer_base + (M)) = htons((Y)))			
300	#define LM_PUT_LONG_AT_MARK(M, X, Y) ((*((u_long *)((X)->outgoing_buffer_base + (M)) = htonl((Y)))			
301	/* (X)->outgoing_buffer_base + (M)) = htonl((Y)))			
302	#else /* SUN4, APOLLO10K, MODELER, HP300 */			
303	#define LM_GET_SHORT(X)			
304	((short *)((X)->incoming_buffer_pointer)++)			
305	/* (short *)((X)->incoming_buffer_pointer)++			
306	#define LM_PUT_SHORT(X, Y)			
307	((short *)((X)->outgoing_buffer_pointer)++ = (Y))			
308	/* (short *)((X)->outgoing_buffer_pointer)++ = (Y)			
309	#define LM_PUT_SHORT_AT_MARK(M, X, Y)			
310	((short *)((X)->outgoing_buffer_base + (u_long)(M)) = (Y))			
311	/* (short *)((X)->outgoing_buffer_base + (u_long)(M)) = (Y)			
312	#define LM_GET_LONG(X)			
313	((long *)((X)->incoming_buffer_pointer)++)			
314	/* (long *)((X)->incoming_buffer_pointer)++			
315	#define LM_PUT_LONG(X, Y)			
316	((long *)((X)->outgoing_buffer_pointer)++ = (Y))			
317	/* (long *)((X)->outgoing_buffer_pointer)++ = (Y)			
318	#define LM_PUT_LONG_AT_MARK(M, X, Y)			
319	((long *)((X)->outgoing_buffer_base + (u_long)(M)) = (Y))			
320	/* (long *)((X)->outgoing_buffer_base + (u_long)(M)) = (Y)			
321	#endif			
322	#endif			
323	/* Machine independent get and put macros */			
324	/* Machine independent get and put macros */			
325	#define LM_CHK_GET_CHAR(X)			
326	((X)->incoming_buffer_pointer < (X)->incoming_message_end ?			
327	LM_GET_CHAR(X) : (char) '\0')			
328	/* (X)->incoming_buffer_pointer < (X)->incoming_message_end ?			
329	#define LM_CHK_GET_LONG(X)			
330	(LM_GET_LONG(X) : (long) MAX_NEGATIVE_LONG)			
331	/* (X)->incoming_buffer_pointer < (X)->incoming_message_end ?			
332	#define LM_CHK_GET_SHORT(X)			
333	((X)->incoming_buffer_pointer+1 < (X)->incoming_message_end ?			
334	LM_GET_SHORT(X) : (short) MAX_NEGATIVE_SHORT)			
335	/* (X)->incoming_buffer_pointer+1 < (X)->incoming_message_end ?			
336	#define LM_CHK_PUT_CHAR(X, Y)			
337	((X)->outgoing_buffer_pointer < (X)->outgoing_buffer_end ?			
338	LM_PUT_CHAR(X, Y) :			
339	(lm_extend_outgoing_buffer(X), LM_PUT_CHAR(X, Y))			
340	/* (X)->outgoing_buffer_pointer < (X)->outgoing_buffer_end ?			
341	#define LM_CHK_PUT_LONG(X, Y)			
342	(LM_PUT_LONG(X, Y) :			
343	(lm_extend_outgoing_buffer(X), LM_PUT_LONG(X, Y))			
344	/* (X)->outgoing_buffer_pointer < (X)->outgoing_buffer_end ?			
345	#define LM_CHK_PUT_SHORT(X, Y)			
346	(LM_PUT_SHORT(X, Y) :			
347	(lm_extend_outgoing_buffer(X), LM_PUT_SHORT(X, Y))			
348	/* (X)->outgoing_buffer_pointer < (X)->outgoing_buffer_end ?			
349	#define LM_CHK_PUT_SHORT(X, Y)			
350	((X)->outgoing_buffer_pointer+1 < (X)->outgoing_buffer_end ?			
351	LM_PUT_SHORT(X, Y) :			
352	(lm_extend_outgoing_buffer(X), LM_PUT_SHORT(X, Y))			
353	/* (X)->outgoing_buffer_pointer+1 < (X)->outgoing_buffer_end ?			
354	extern CONNECTION *lm_global_conn_ptr;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/nvsram.h	DATE 5/23/89	PAGE # 1/58
LINE #		HEADER TEXT		
1	2	/* SC3 ID: svram.h rev 1.1, 4/25/89 at 16:51:09 */		
3	4	/*		
5	6	/* 16-1000 Host Utility include file		
7	8	/* copyright 1988, Logic Modeling Systems, Inc.		
9	10	/* ethernet packet type is 0000 hex for Unix and other systems		
11	12	/* that support NW/IP. For VMS we need to find a number		
13	14	/* which won't collide with any other protocols.		
15	16	/*		
17	18	/* host_machine_type 1 = IBM		
19	20	/* 2 = APOLLO		
21	22	/* 3 = VAX/VMS		
23	24	/* 4 = IBM/PC		
25	26	/*		
27	28	/* autoboot 1 = modular should autoboot		
29	30	/* 0 = modular should manual boot		
31	32	/*		
33	34	/* diagnostics 1 = diagnostic		
35	36	/* 0 = no diagnostic		
37	38	/*		
39	40	/* this needs to be differentiated because PCs may not have an		
41	42	/* actual autoboot facility....		
43	44	/*		
45	46	/* This is the current EPROM_NVRAM_VERSION		
47	48	/* If it does not match NVSRAM, we need either new EPROM or		
49	50	/* new host utility.		
51	52	/* if EPROM_NVRAM_VERSION > BOOT_STRUCT.version:		
53	54	/* new host utility.		
55	56	/* if EPROM_NVRAM_VERSION < BOOT_STRUCT.version:		
57	58	/* new EPROM.		
59	60	/*		
61	62	#define EPROM_NVRAM_VERSION 0x0002		
63	64	typedef struct {		
65	66	unsigned char len[4];		
67	68	unsigned short version;		
69	70	unsigned short ethernet_packet_type;		
71	72	unsigned short internet_socket_number;		
73	74	unsigned char host_ethernet_address[6];		
75	76	host_internet_address;		
77	78	modem_internet_address;		
79	80	host_machine_type;		
81	82	autoboot;		
83	84	diagnostics;		
85	86	};		
87	88	/* WARNING: This structure must be an even number of		
89	90	/* bytes on VMS, otherwise the sizeof		
91	92	/* function produces "incorrect" results.		
93	94	/* VMS is aligned to bytes, Unix to words.		
95	96	/*		
97	98	} BOOT_STRUCT;		
99	100	/*		
101	102	/* The following must match the boot structure		
103	104	/*		
105	106	#define BSIC (unsigned long) 0		
107	108	#define EVER (unsigned long) (BSIC + 4*4)		
109	110	#define RESET_TYPE (unsigned long) (EVER + 2*4)		
111	112	#define REPORT (unsigned long) (RESET_TYPE + 2*4)		
113	114	#define REDNET (unsigned long) (REPORT + 2*4)		
115	116	#define BRINET (unsigned long) (REDNET + 4*4)		
117	118	#define BRINET (unsigned long) (BRINET + 4*4)		
119	120	#define BROST (unsigned long) (BRINET + 4*4)		
121	122	#define BASTO (unsigned long) (BROST + 1*4)		
123	124	#define DIAGNOSTICS (unsigned long) (BASTO + 1*4)		
125	126	#define BOOT (unsigned long) (DIAGNOSTICS + 1*4)		
127	128	#define BSIC (unsigned long) 0		
129	130	#define SIZEOF_BSIC (unsigned long) 4		
131	132	#define SIZEOF_EVER (unsigned long) 2		
133	134	#define SIZEOF_RESET_TYPE (unsigned long) 2		
135	136	#define SIZEOF_REPORT (unsigned long) 2		
137	138	#define SIZEOF_REDNET (unsigned long) 6		
139	140	#define SIZEOF_BRINET (unsigned long) 4		
141	142	#define SIZEOF_BRINET (unsigned long) 4		
143	144	#define SIZEOF_BROST (unsigned long) 1		
145	146	#define SIZEOF_BASTO (unsigned long) 1		
147	148	#define SIZEOF_DIAGNOSTICS (unsigned long) 1		
149	150	#define SIZEOF_BOOT (unsigned long) sizeof(BOOT_STRUCT)		
151	152	#define MODELER_STATE (unsigned long)((sizeof(BOOT_STRUCT))*4)		
153	154	#define MODELER_RESET (unsigned long)(MODELER_STATE + 1*4)		
155	156	/*		
157	158	/* band rate selection		
159	160	/*		
161	162	#define BAUD_RATEA (unsigned long) (MODELER_RESET + 1*4)		
163	164	#define BAUD_RATEB (unsigned long) (BAUD_RATEA + 1*4)		
165	166	#define CONSOLE_BAUD BAUD_RATEA		
167	168	#define MODEM_BAUD BAUD_RATEB		
169	170	/*		
171	172	/* Gab insertion count followed by		
173	174	/* number of times pattern played		
175	176	/*		
177	178	#define RUNTIME_S1 (unsigned long)(BAUD_RATEB + 1*4)		
179	180	/*		
181	182	/* bus error address		
183	184	/* and special status word		
185	186	/*		
187	188	#define BUS_ADDR (unsigned long)(RUNTIME_S1 + sizeof(RUNTIME_STAT_STRUCT)*4)		
189	190	#define ISW_REG (unsigned long)(BUS_ADDR + 4*4)		
191	192	/*		
193	194	/* clock board, PMM, PSL, etc.		
195	196	/*		
197	198	#define HARDWARE_ERROR (unsigned long)(ISW_REG + 2*4)		
199	200	/*		
201	202	/* CPU parity error		
203	204	/*		
205	206	#define PARITY_ERR (unsigned long)(HARDWARE_ERROR + (sizeof(LM_HARDWARE_ERROR)*4))		
207	208	/*		
209	210	/* reason for reset, see defines at the end of file		
211	212	/*		
213	214	/*		
215	216	/* network timeout		
217	218	/*		
219	220	/*		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/nvsram.h	DATE 5/23/89 TIME 1:20:22 pm	PAGE # 2/59
LINE #	HEADER TEXT			
121	#define NETWORK_TIMEOUT (unsigned long)(PARITY_ERR + 4*4)			
122	/*			
123	** Configuration errors			
124	*/			
125	#define CONFIG_ERROR (unsigned long)(NETWORK_TIMEOUT + 2*4)			
126	#define PAC_PAN_ERROR (unsigned long)(CONFIG_ERROR + sizeof(CONFIGURATION_ERRORS) * 4) /* (16 bytes * 4) */			
127	#define LANCE_CIR_REG (unsigned long)(PAC_PAN_ERROR + 4*4)			
128	#define ERROR_VECTOR (unsigned long)(LANCE_CIR_REG + 2*4)			
129	#define PASSWORD (unsigned long)(ERROR_VECTOR + 2*4)			
130	/*			
131	#define CPU_SKIP_MENU (unsigned long)(0x7fd * 4)			
132	#define CPU_TEST_BYTE (unsigned long)(0x7fe * 4)			
133	#define CHECKSUM (unsigned long)(0x7ff * 4)			
134	/*			
135	#define SIZEOF_RUNTIME_STAT (unsigned long) (sizeof(RUNTIME_STAT_STRUCT))			
136	#define SIZEOF_BAUD_RATEA (unsigned long) 1			
137	#define SIZEOF_BAUD_RATEB (unsigned long) 1			
138	#define SIZEOF_CONSOLE_BAUD (unsigned long) 1			
139	#define SIZEOF_MODEM_BAUD (unsigned long) 1			
140	#define SIZEOF_BUS_ADDR (unsigned long) 4			
141	#define SIZEOF_ISM_REG (unsigned long) 2			
142	#define SIZEOF_HARDWARE_ERROR (unsigned long) (sizeof(LM_HARDWARE_ERROR))			
143	#define SIZEOF_PARITY_ERR (unsigned long) 4			
144	#define SIZEOF_MODELER_STATE (unsigned long) 1			
145	#define SIZEOF_MODELER_RESET (unsigned long) 1			
146	#define SIZEOF_NETWORK_TIMEOUT (unsigned long) 2			
147	#define SIZEOF_CONFIG_ERROR (unsigned long) (sizeof(CONFIGURATION_ERRORS))			
148	#define SIZEOF_PAC_PAN_ERROR (unsigned long) 16			
149	#define SIZEOF_LANCE_CIR_REG (unsigned long) 2			
150	#define SIZEOF_ERROR_VECTOR (unsigned long) 2			
151	#define SIZEOF_PASSWORD (unsigned long) 16			
152	#define SIZEOF_CPU_SKIP_MENU (unsigned long) 1			
153	#define SIZEOF_CPU_TEST_BYTE (unsigned long) 1			
154	#define SIZEOF_CHECKSUM (unsigned long) 1			
155	/*			
156	** Boot options			
157	*/			
158	#define AUTOSBOOT 1			
159	#define MANUALBOOT 0			
160	#define APPLICATION_BOOT 0			
161	#define DIAGNOSTIC_BOOT 1			
162	/*			
163	#define SUN			
164	#define LM_BOOT_PACKET_TYPE 0x0800			
165	#define LM_BOOT_MACHINE 1			
166	#endif			
167	/*			
168	#define APOLLO			
169	#define LM_BOOT_PACKET_TYPE 0x0800			
170	#define LM_BOOT_MACHINE 2			
171	#endif			
172	/*			
173	#define VMS			
174	#define LM_BOOT_PACKET_TYPE 0x07ff /* FILIT get a better number */			
175	#define LM_BOOT_MACHINE 3			
176	#endif			
177	/*			
178	** Modeler states			
179	*/			
180	#define SHUTDOWN (char)2 /* state */			
181	#define REBOOT (char)3 /* state */			
182	#define DELAY_REBOOT (char)4 /* state */			
183	#define BOOTING (char)5 /* state */			
184	#define BOOTING (char)6 /* state */			
185	#define BOOTING (char)7 /* state */			
186	#define BOOTING (char)8 /* state */			
187	#define BOOTING (char)9 /* state */			
188	#define MODELER_RUNNING (char)31 /* state */			
189	#define MODELER_CALIBRATING (char)32 /* state */			
190	#define RUNNING_LOOPMODE (char)33 /* state */			
191	#define RUNNING_DIAG (char)34 /* state */			
192	/*			
193	** Modeler reset conditions			
194	*/			
195	#define PARITY_ERROR (char)1 /* Reset condition */			
196	#define BACKPLANE_ERROR (char)4 /* Reset condition */			
197	#define BUS_ERROR (char)5 /* Reset condition */			
198	#define LOST_POWER (char)9 /* Reset condition */			
199	#define SUICIDE (char)10 /* Reset condition */			
200	#define SUICIDE1 (char)11 /* Reset condition */			
201	#define SUICIDE2 (char)12 /* Reset condition */			
202	#define SUICIDE3 (char)13 /* Reset condition */			
203	#define SUICIDE4 (char)14 /* Reset condition */			
204	#define SUICIDE5 (char)15 /* Reset condition */			
205	#define SUICIDE6 (char)16 /* Reset condition */			
206	#define SUICIDE7 (char)17 /* Reset condition */			
207	#define SUICIDE8 (char)18 /* Reset condition */			
208	#define SUICIDE9 (char)19 /* Reset condition */			
209	#define SUICIDE10 (char)20 /* Reset condition */			
210	#define SUICIDE11 (char)21 /* Reset condition */			
211	#define SUICIDE12 (char)22 /* Reset condition */			
212	#define SUICIDE13 (char)23 /* Reset condition */			
213	#define SUICIDE14 (char)24 /* Reset condition */			
214	#define SUICIDE15 (char)25 /* Reset condition */			
215	#define SUICIDE16 (char)26 /* Reset condition */			
216	#define SUICIDE17 (char)27 /* Reset condition */			
217	/*			
218	** Primary values for CPU_SKIP_MENU			
219	*/			
220	#define CPU_SKIP_MENU_YES (char)174			
221	/*			
222	** This is the NVSRAM, as a data structure			
223	*/			
224	typedef struct {			
225	BOOT_STRUCT boot; /* EPROM version dependent */			
226	unsigned char modeler_state; /* EPROM version dependent */			
227	unsigned char modeler_reset; /* EPROM version dependent */			
228	unsigned char baud_ratea; /* EPROM version dependent */			
229	unsigned char baud_rateb; /* EPROM version dependent */			
230	RUNTIME_STAT_STRUCT rtme_structure; /* EPROM version dependent */			
231	unsigned long bus_addr;			
232	unsigned short spec_status_word;			
233	LM_HARDWARE_ERROR lm_hardware_error;			
234	unsigned long parity_error;			
235	unsigned short network_timeout;			
236	CONFIGURATION_ERRORS config_errors;			
237	unsigned long pac_pan_errors(4);			
238	unsigned short lance_cir;			
239	unsigned short error_vector;			
240	unsigned char password(16);			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/nvsram.h	DATE 5/23/89	PAGE # 3/60
LINE #	HEADER TEXT			
241)NVSRAM_ORGANIZATION,			
242				
243	/* This is needed to figure out how much is reserved in between			
244	/* the above struct, and the checksum and Modeler's diag.			
245	*/			
246				
247	typedef struct {			
248	NVSRAM_ORGANIZATION nvsram,			
249	unsigned char reserved[0x1000 - sizeof(NVSRAM_ORGANIZATION)]; /* where 1000 is the size of nvsram */			
250	unsigned char cpu_skip_word,			
251	unsigned char cpu_test_byte,			
252	unsigned char checksum,			
253	} NVSRAM_ORG;			
254				
255	extern char modeler_state,			

Copyright 1989 Logic Modeling Systems	HEADER FILE	DATE	PAGE #
	include/pac.h	5/23/89	1/61
		TIME	1:20:22 pm

LINE #	HEADER TEXT
1	/* SOCI_ID: pac.h rev 3.1, 4/24/89 at 07:34:28 */
2	/*
3
4
5
6
7
8
9
10	#define EMPTY_124_BYTES unsigned \
11	:32, :32, :32, :32, :32, :32, :32, :32, :32, \
12	:32, :32, :32, :32, :32, :32, :32, :32, :32, \
13	:32, :32, :32, :32, :32, :32, :32, :32, :32, :32;
14	#define EMPTY_SIZE 128
15	/*
16	* There are from one to four PACs in the LM-1000, one for each of
17	* the four lanes (A, B, C, and D). Each PAC can have from one to
18	* four PAM (pattern memory) boards plugged into it.
19	* Pattern memory is organized in three banks.
20	* Bank 0 is offset 00000000 from the lane address.
21	* Bank 1 is offset 00000000 from Bank 0.
22	* Bank 2 is offset 00000000 from Bank 1.
23	*/
24	#define PAT_MEM_BANK (u_long)0x00000000 /* 64 Kbytes */
25	#define BANK_0 (u_long)00000000
26	#define BANK_1 (BANK_0 + PAT_MEM_BANK)
27	#define BANK_2 (BANK_1 + PAT_MEM_BANK)
28	/*
29	* The PAC Link Table is offset by 00000000 from the lane address.
30	* The current size of the link table is 32Kx16, on long word boundaries.
31	*/
32	#define LINK_OFFSET (u_long)0x00000000
33	#define LINK_SIZE (u_long)000020000
34	/*
35	* The following is a data structure for all PAC/PAM registers.
36	* The PAC/PAM registers are offset by 0x0C100000 from the lane address.
37	* All read/write registers are cleared on backplane reset (i.e., Power-on).
38	* The PAC asserts ERROR when any one of the six error conditions is true.
39	*/
40	#define PAC_REG_OFFSET (u_long)0xc100000
41	typedef struct
42	{
43	/* PAC ID PROM */
44	/* offset: 0x0 = 0 */
45	#define PAC_ID PAC_REG_OFFSET
46	/* The PAC ID PROM is a byte-wide resource on 12 bit boundaries. */
47	struct
48	{
49	unsigned :24, data:8;
50	} id_prom[32];
51	/* PAC/PAM COUNT REGISTER */
52	/* offset: 0x0 = 128 = 4 + 124 + 0 */
53	#define PAM_COUNT (PAC_ID + (u_long)0x0)
54	/* Read-only, 4-bit register */
55	unsigned :28,
56	pam_count:4;
57	/* 4-bit code indicating number of pams */
58	/* stacked onto the PAC */
59	/*
60	pam_count:0;
61	/*
62	#define ZERO_PAMS 0x0
63	#define ONE_PAM 0x1
64	#define TWO_PAMS 0x2
65	#define THREE_PAMS 0x3
66	#define FOUR_PAMS 0x4
67	/*
68	EMPTY_124_BYTES
69	/*
70	PAC CONFIGURATION REGISTER */
71	/* offset: 0x100 = 256 = 4 + 124 + 128 */
72	#define CONFIG_OFFSET (PAM_COUNT + EMPTY_SIZE)
73	unsigned :24,
74	high_word_parity:1, /* parity setting on the high word */
75	low_word_parity:1, /* parity setting on the low word */
76	configuration:6;
77	/*
78	word_parity:0;
79	/*
80	#define SET_ODD_PARITY 1
81	#define SET_EVEN_PARITY 0
82	/*
83	/* configuration pseudo code
84	if (pam_count != ONE_PAM)
85	{
86	configuration = (total number of patterns / 128K) - 1;
87	}
88	else
89	{
90	switch(number of patterns in PAM)
91	{
92	128K: configuration = 0; break;
93	512K: configuration = 41; break;
94	2048K: configuration = 63; break;
95	}
96	}
97	/*
98	EMPTY_124_BYTES
99	/*
100	PAC BRANCH ADDRESS REGISTER */
101	/* offset: 0x180 = 384 = 4 + 124 + 256 */
102	#define BRANCH_OFFSET (CONFIG_OFFSET + EMPTY_SIZE)
103	unsigned :16,
104	branch_address:15;
105	/* contains block address of first
106	/* block in pattern sequence */
107	/*
108	EMPTY_124_BYTES
109	/*
110	PAC CLOCK SPEED REGISTER */
111	/* offset: 0x200 = 512 = 4 + 124 + 384 */
112	#define CLOCK_OFFSET (BRANCH_OFFSET + EMPTY_SIZE)
113	unsigned :31,
114	clock_speed:1;
115	/* clock speed setting (1MHz or >1MHz) */
116	/*
117	/* clock speed */
118	#define BELOW_1MHZ 0
119	#define ABOVE_1MHZ 1

Copyright 1989 Logic Modeling Systems		HEADER FILE include/pac.h	DATE 5/23/89	PAGE # 2/62
LINE #		HEADER TEXT		
121		EMPTY_124_BYTES		
122		/* PAC ERROR REGISTERS */		
123		/* offset: 0x200 = 0 - 4 + 124 + 312 */		
124		#define ERROR_OFFSET (LOCK_OFFSET + EMPTY_SIZE)		
125		/* Read only 4-bit registers. All bits are active high. */		
126		/* Register is cleared by writing any value to this address */		
127		/* Fatal refresh error */		
128		/* Fatal Request State Machine error */		
129		/* Fatal Pattern State Machine error */		
130		/* Fatal Pattern presentation control word */		
131		/* parity error */		
132		/* Pattern error */		
133		/* Pattern error */		
134		/* Pattern error */		
135		EMPTY_124_BYTES		
136		/* PAC BLOCK OFFSET REGISTER */		
137		/* offset: 0x300 = 748 - 4 + 124 + 540 */		
138		#define BLOCK_OFFSET (ERROR_OFFSET + EMPTY_SIZE)		
139		/* Pattern offset within a pattern block */		
140		/* on presentation halt due to an error */		
141		/* There is some, as yet undetermined, */		
142		/* latency in this address that must be */		
143		/* accounted for. Must read before */		
144		/* errors are cleared. */		
145		EMPTY_124_BYTES		
146		/* PAC PARITY ERROR ADDRESS REGISTER */		
147		/* offset: 0x380 = 896 - 4 + 124 + 760 */		
148		#define PARITY_OFFSET (BLOCK_OFFSET + EMPTY_SIZE)		
149		/* error on high word parity */		
150		/* error on low word parity */		
151		/* both could be set. */		
152		/* constrains A[27:2] on CPU read */		
153		/* success parity error. */		
154		/* Must read before clearing errors. */		
155		EMPTY_124_BYTES		
156		/* PAC PRESENT REGISTERS and ID PROMS */		
157		/* offset: 0x400 = 1024 - 4 + 124 + 896 */		
158		#define PAM_OFFSET (PARITY_OFFSET + EMPTY_SIZE)		
159		#define PAM_ID (PAM_OFFSET + EMPTY_SIZE)		
160		/* Only pam_register[0] contains valid data. */		
161		/* struct */		
162		/* indication that a PAM is present. */		
163		/* 124 bytes of hole. */		
164		/* The PAM ID PROM is a byte wide memory, on 32 bit boundaries. */		
165		/* struct */		
166		/* data:8, */		
167		/* id_prom[32], */		
168		/* pam_register[4], */		
169		/* pam_register[1].id_prom[0].data */		
170		#define PAM_TYPE id_prom[0].data		
171		#define PAM_128K 0x05		
172		#define PAM_512K 0x07		
173		#define PAM_2M 0x13		
174		/* pam_register[1].present */		
175		#define PAM_PRESENT 0		
176		#define PAM_NOT_PRESENT 1		
177		/* PAC, */		
178		/* PAC, */		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/pac_def.h	DATE 5/23/89 TIME 1:20:22 pm	PAGE # 1/63
LINE #	HEADER TEXT			
1	/* SCCS ID: pac_def.h rev 3.1, 4/24/89 at 07:34:30 */			
2	/*			
3	pac_def.h			
4	/*			
5	Definitions, macros, and structures			
6	used in PAC diagnostics			
7	/*			
8	/* PAC information table */			
9	typedef struct			
10	{			
11	int exists; /* TRUE or FALSE */			
12	u_long lane_offset;			
13	int num_pams;			
14	int pam_size[4];			
15	int num_blocks;			
16	int num_patterns;			
17	} PAC_INFO;			
18	/*			
19	/* Pattern word structure			
20	/*			
21	pattern bits 79..64 = PAT_WORD.pattern.data[0] \ PAT_WORD.mem_bank[0]			
22	pattern bits 63..48 = PAT_WORD.pattern.data[1] \			
23	/*			
24	pattern bits 47..32 = PAT_WORD.pattern.data[2] \ PAT_WORD.mem_bank[1]			
25	pattern bits 31..16 = PAT_WORD.pattern.data[3] \			
26	/*			
27	pattern bits 15..0 = PAT_WORD.pattern.data[4] \ PAT_WORD.mem_bank[2]			
28	/*			
29	control bits 15..0 = PAT_WORD.pattern.control			
30	/*			
31	/*			
32	typedef union			
33	{			
34	u_long mem_bank[3];			
35	struct			
36	{			
37	u_short data[5];			
38	unsigned backplane_spare : 2;			
39	unsigned pac_spare : 3;			
40	unsigned branch : 2;			
41	unsigned stop : 1;			
42	unsigned word : 1;			
43	unsigned pel_control : 3;			
44	unsigned pel_address : 4;			
45	} pattern;			
46	} PAT_WORD;			
47	/* PAC defines */			
48	#define PATTERNS_IN_2M (2 << 20)			
49	#define PATTERNS_IN_512K (512 << 10)			
50	#define PATTERNS_IN_128K (128 << 10)			
51	#define BLOCK_SIZE 256			
52	#define BLKS_IN_2M (PATTERNS_IN_2M / BLOCK_SIZE)			
53	#define BLKS_IN_512K (PATTERNS_IN_512K / BLOCK_SIZE)			
54	#define BLKS_IN_128K (PATTERNS_IN_128K / BLOCK_SIZE)			
55	#define BRANCH_LATENCY 27			
56	#define MAX_BRANCHES ((BLOCK_SIZE - (BRANCH_LATENCY - 1)) / 2)			
57	#define STOP_LATENCY 4			
58	#define STOP 0x0100			
59	#define BRANCH_ALWAYS 0x0200			
60	#define BRANCH_FALLING 0x0400			
61	#define BRANCH_RISING 0x0800			
62	#define TRIGGER 0x0080			
63	#define STOP_MASK (u_long)((STOP BRANCH_RISING TRIGGER))			
64	#define PAM_ID_SPACE 0x0100			
65	#define LOOP_MODE 1			
66	#define STOP_MODE 0			
67	#define FIRST_BLOCK 0			
68	#define PAC_MAX_PERIOD 7900 /* 128 kHz (actually 126.582 kHz) */			
69	#define PAC_MIN_PERIOD 38 /* 15 MHz (actually 26.3 MHz) */			
70	/* CPU access (read/write) macros */			
71	/*			
72	/* Macro Write_long(X, Y)			
73	/*			
74	INPUT: X = pattern memory long address			
75	Y = 32-bit data word			
76	OUTPUT: none			
77	DESCRIPTION: Performs long word pattern memory write			
78	operation.			
79	/*			
80	#define Write_long(X, Y) (*(u_long *) (X) = (Y))			
81	/*			
82	/* Macro Write_word(X, Y)			
83	/*			
84	INPUT: X = pattern memory word address			
85	Y = 16-bit data word			
86	OUTPUT: none			
87	DESCRIPTION: Performs word pattern memory write			
88	operation.			
89	/*			
90	#define Write_word(X, Y) (*(u_short *) (X) = (Y))			
91	/*			
92	/* Macro Read_long(X)			
93	/*			
94	INPUT: X = pattern memory long address			
95	OUTPUT: long data value			
96	DESCRIPTION: Performs long word pattern memory read			
97	operation.			
98	/*			
99	#define Read_long(X) (*(u_long *) (X))			
100	/*			
101	/* Macro Read_word(X)			
102	/*			
103	INPUT: X = pattern memory long address			
104	OUTPUT: short data value			
105	DESCRIPTION: Performs word pattern memory read			
106	operation.			
107	/*			
108	#define Read_word(X) (*(u_short *) (X))			
109	/*			
110	/* Other PAC/PAM macros */			
111	/*			
112	/* Macro Clear_pac_errors(X)			
113	/*			
114	INPUT: X = lane number (0, 1, 2, or 3)			
115	/*			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/pac_def.h	DATE 5/23/89	PAGE # 2/64
LINE #		HEADER TEXT		
121	/*	OUTPUT: none		
122	DESCRIPTION:	Clears PAC error registers in specified lane.		
123	/*			
124	#define	Clear_pac_errors(X) (pacptr(X)->parity_error = 0)		
125	/*			
126	/* Random number generation definitions */			
127	/*			
128	#define	TAP_MASK ((1 << 0) (1 << 1) (1 << 2) (1 << 4) (1 << 6))		
129	#define	MS_TAP (1 << 31)		
130	/*			
131	macro	Pac_get_random(X)		
132	/*			
133	INPUT: X = 32-bit random number			
134	OUTPUT: next 32-bit random number			
135	DESCRIPTION:	Generates next 32-bit pseudorandom number		
136	based on current random number.			
137	/*			
138	#define	Pac_get_random(X) (((X)&MS_TAP)?(((X)^TAP_MASK)<<1) 1:(X)<<1)		
139	/*			
140	/*			
141	macro	Address_to_pattern(X)		
142	/*			
143	INPUT: X = 32-bit pattern memory address			
144	OUTPUT: 32-bit pattern number			
145	DESCRIPTION:	Converts address to pattern number.		
146	/*			
147	#define	Address_to_pattern(X) ((int)((X)>>2) & ~(0x3f << 24))		
148	/*			
149	/*			
150	macro	Pattern_to_address(X,Y,Z)		
151	/*			
152	INPUT: X = lane number			
153	Y = bank number			
154	Z = 32-bit pattern number			
155	OUTPUT: 32-bit pattern memory address			
156	DESCRIPTION:	Converts pattern number to address.		
157	/*			
158	#define	Pattern_to_address(X,Y,Z) (pac[(X)].lane_offset + \		
159	((Y) * PAT_MEM_BANK) + ((Z) << 2))			
160	/*			
161	/*			
162	macro	Pan_base_address(X,Y)		
163	/*			
164	INPUT: X = lane number			
165	Y = pan number			
166	OUTPUT: 32-bit pattern memory address			
167	DESCRIPTION:	Computes pattern memory base address (bank 0)		
168	for specified Pattern Memory Bank.			
169	/*			
170	#define	Pan_base_address(X,Y) (Pattern_to_address((X), 0, \		
171	(pac_get_first_pattern_no(X), (Y))))			
172	/*			
173	/*			
174	macro	Pac_set_parity(X,Y)		
175	/*			
176	INPUT: X = lane number			
177	Y = SET_ODD_PARITY or SET_EVEN_PARITY			
178	OUTPUT: none			
179	DESCRIPTION:	Configures both the high and the low		
180	word parity generators to compute the specified			
181	parity. Done for PAC in lane X.			
182	/*			
183	#define	Pac_set_parity(X,Y) (pacptr[(X)]->high_word_parity = (Y) \		
184	pacptr[(X)]->low_word_parity = (Y))			
185	/*			
186	/*			
187	macro	Pac_parity_address(X)		
188	/*			
189	INPUT: X = lane number			
190	OUTPUT: none			
191	DESCRIPTION:	Computes address of parity error based		
192	on value latched in parity error address register.			
193	/*			
194	#define	Pac_parity_address(X) (pac[(X)].lane_offset + \		
195	(pacptr[(X)]->parity_error_address << 2))			
196	/*			
197	/*			
198	macro	Pac_clock_speed(X,Y)		
199	/*			
200	INPUT: X = lane number			
201	Y = clock period in nanoseconds			
202	OUTPUT: none			
203	DESCRIPTION:	Sets Pattern Controller clock speed register		
204	based on clock period.			
205	/*			
206	#define	Pac_clock_speed(X,Y) (pacptr[(X)]->clock_speed = ((Y) > 1000) ? \		
207	BELOW_LIMIT : ABOVE_LIMIT)			
208	/*			
209	/*			
210	#define	MIN_TIMEOUT 10 /* Min timeout of 10 ms */		
211	/*			
212	macro	Timeout(X)		
213	/*			
214	INPUT: X = expected duration of pattern play in ms			
215	OUTPUT: none			
216	DESCRIPTION:	Computes pattern play time (expected time multiplied		
217	by two, or min timeout).			
218	/*			
219	#define	Timeout(X) (((X) << 1) > MIN_TIMEOUT) ? (X) << 1 : MIN_TIMEOUT)		
220	/*			
221	/*			

[illegible]

Copyright 1989 Logic Modeling Systems		HEADER FILE include/protans.h	DATE 5/23/89	PAGE # 1/66
LINE #	HEADER TEXT			
1	/* SOCS_ID: protans.h Rev 3.1.1, 4/24/89 at 07:34:37 */			
2	/* BEGIN CONST_SECTION */			
3	#define READ_ANS 3			
4	#define WRITE_ANS 5			
5	#define CREATE_DEF_ANS 7			
6	#define CREATE_INSTANCE_ANS 9			
7	#define CREATE_FAULT_ANS 11			
8	#define RELEASE_DEF_ANS 13			
9	#define RELEASE_INSTANCE_ANS 15			
10	#define RELEASE_FAULT_ANS 17			
11	#define EVAL_ANS 19			
12	#define SAVE_DEF_ANS 21			
13	#define SAVE_PTRN_ANS 23			
14	#define SAVE_PTRN_COST_ANS 25			
15	#define RESTORE_INST_ANS 27			
16	#define RESTORE_PTRN_ANS 29			
17	#define PTRN_HIST_ANS 31			
18	#define IMG_MODELER_ANS 33			
19	#define IMG_USER_LIST_ANS 35			
20	#define IMG_USER_ANS 37			
21	#define IMG_LANE_ANS 39			
22	#define IMG_PAN_ANS 41			
23	#define IMG_PEL_ANS 43			
24	#define IMG_DEVICE_LIST_ANS 45			
25	#define IMG_DEVICE_NAME_ANS 47			
26	#define IMG_DEVICE_ANS 49			
27	#define IMG_DAB_ANS 51			
28	#define IMG_DAB_LOC_ANS 53			
29	#define IMG_INSTANCE_ANS 55			
30	#define IMG_FAULT_ANS 57			
31	#define IMG_AVAIL_PTRN_ANS 59			
32	#define THEASUREMENT_ANS 61			
33	#define EXTRACT_DEF_ANS 63			
34	#define LOOP_PTRN_ANS 65			
35	#define RESET_INST_ANS 67			
36	#define TEST_NETWORK_ANS 69			
37	#define ABORT_ANS 71			
38	#define BEGIN_SESSION_ANS 73			
39	#define LABEL_DAB_ANS 75			
40	#define EVAL_CONTROL_ANS 77			
41	#define REBOOT_ANS 79			
42	#define SHUTDOWN_ANS 81			
43	#define CHECK_DAMAGE_ANS 83			
44	#define SAVE_DEF_COST_ANS 85			
45	#define PASSWORD_ANS 87			
46	#define NO_SUCH_ANS 89			
47	/* END CONST_SECTION */			
48	#define DELAY_TABLE_COMPOSITE 0			
49	#define DELAY_TABLE 1			
50	#define MEASURED 2			
51	typedef struct {			
52	u_short pin_number;			
53	u_char pin_value;			
54	u_short event_pin_number;			
55	u_char delay_type; /* DELAY_TABLE_COMPOSITE DELAY_TABLE MEASURED */			
56	union {			
57	struct {			
58	u_long min_delay; /* delay_type == DELAY_TABLE_COMPOSITE */			
59	u_long typ_delay;			
60	u_long max_delay;			
61	OPC_delay_table_composite;			
62	long delay_table_index; /* delay_type == DELAY_TABLE */			
63	struct {			
64	u_long min_delay; /* delay_type == MEASURED */			
65	u_long max_delay;			
66	OPC_measured;			
67	} OPC_union;			
68	} OUTPUT_PIN_CHANGE;			
69	typedef struct {			
70	u_char error_type;			
71	char error_string[1];			
72	} ERROR;			
73	typedef struct {			
74	u_long error_count;			
75	union {			
76	ERROR error_el[1];			
77	struct {			
78	u_short data_id;			
79	u_char use_default;			
80	u_char report_name;			
81	MIN_TTP_MAX default_delay;			
82	u_long ntype_count;			
83	MIN_TTP_MAX delay_table[1];			
84	u_long pin_count;			
85	u_short pin_number;			
86	u_short pin_class; /* DATA, EVAL, STORE */			
87	u_short pin_type; /* NONE, IM, OUT, IO */			
88	char pin_name[1];			
89	char pin_number_str[1];			
90	char pin_alias[1];			
91	} CDA_struct;			
92	} CDA_union;			
93	} creates_def_ans;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/protans.h	DATE 5/23/89	PAGE # 2/67
LINE #	HEADER TEXT			
121				
122	typedef struct {			
123	u_short pin_number;			
124	u_char pin_value;			
125	} INIT_PIN_VALUE;			
126				
127	typedef struct {			
128	u_long error_count;			
129	union {			
130	ERROR error_el[1];			
131	struct {			
132	u_short instance_id;			
133	u_short output_count;			
134	INIT_PIN_VALUE init_pin_value[1];			
135	} CIA_struct;			
136	} CIA_union;			
137	} create_instance_ana;			
138				
139	typedef struct {			
140	u_long error_count;			
141	union {			
142	ERROR error_el[1];			
143	u_short fault_id;			
144	} CFA_union;			
145	} create_fault_ana;			
146				
147	typedef struct {			
148	u_long error_count;			
149	ERROR error_el[1];			
150	} release_def_ana;			
151				
152	typedef struct {			
153	u_long error_count;			
154	ERROR error_el[1];			
155	} release_instance_ana;			
156				
157	typedef struct {			
158	u_long error_count;			
159	ERROR error_el[1];			
160	} release_fault_ana;			
161				
162	typedef struct {			
163	u_long error_count;			
164	union {			
165	ERROR error_el[1];			
166	struct {			
167	u_short output_count;			
168	OUTPUT_PIN_CHANGE output_list[1];			
169	} EA_struct;			
170	} EA_union;			
171	} eval_ana;			
172				
173	typedef struct {			
174	u_long error_count;			
175	union {			
176	ERROR error_el[1];			
177	struct {			
178	u_long device_pattern_count;			
179	u_char definition[1];			
180	} SDA_struct;			
181	} SDA_union;			
182	} save_def_ana;			
183				
184	typedef struct {			
185	u_long error_count;			
186	union {			
187	ERROR error_el[1];			
188	struct {			
189	u_char definition[1];			
190	} SDCA_struct;			
191	} SDCA_union;			
192	} save_def_cost_ana;			
193				
194	typedef struct {			
195	u_long error_count;			
196	union {			
197	ERROR error_el[1];			
198	struct {			
199	u_char unit_count;			
200	u_long total_device_ptrn_count;			
201	/* % of actual user's ptrns */			
202	u_long common_ptrn_count;			
203	/* % of common device ptrns for fault seq */			
204	u_char check_input_x_count;			
205	u_char first_eval;			
206	u_char sample_count;			
207	u_long evaluation_count;			
208	u_char purge_on_next_eval;			
209	u_char has_history;			
210	u_char use_2_bit_per_pin;			
211	u_char enable_timing_mask;			
212	u_short pin_count;			
213	struct {			
214	u_short pin_number;			
215	u_char old_raw;			
216	u_char old_filtered;			
217	u_char new_raw;			
218	u_char new_filtered;			
219	u_long sin_time;			
220	} pin_info_array[1];			
221	} SFA_struct;			
222	} SFA_union;			
223	} save_ptrn_ana;			
224				
225	PTN BITS LONGWORD ptrn_loaded;			
226	PTN BITS LONGWORD bmdmb_loaded;			
227	PTN BITS LONGWORD lcychnb_loaded;			
228	PTN BITS LONGWORD last_consistent_set;			
229	PTN BITS LONGWORD sin_pin_value_data;			
230	PTN BITS LONGWORD sin_pin_value_hiz;			
231	PTN BITS LONGWORD sin_pin_value_unk;			
232	PTN BITS LONGWORD sin_pin_value_soft;			
233	PTN BITS LONGWORD last_sample_val_data;			
234	PTN BITS LONGWORD last_sample_val_hiz;			
235	PTN BITS LONGWORD last_sample_val_unk;			
236	PTN BITS LONGWORD last_sample_val_soft;			
237	PTN BITS LONGWORD patterns;			
238	} SPA_struct;			
239	} SPA_union;			
240	} save_ptrn_ana;			
241				
242	typedef struct {			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/protans.h	5/23/89	3/68
			TIME 1:20:23 pm	
LINE #	HEADER TEXT			
241	u_long error_count;			
242	union {			
243	ERROR error_el[1];			
244	struct {			
245	u_long pattern_count_follows;			
246	} SPCA_struct;			
247	} SPCA_union;			
248	} save_ptr_struct;			
249	typedef struct {			
250	u_long error_count;			
251	ERROR error_el[1];			
252	} restore_list_struct;			
253	typedef struct {			
254	u_long error_count;			
255	ERROR error_el[1];			
256	} restore_ptr_struct;			
257	typedef struct {			
258	u_long error_count;			
259	ERROR error_el[1];			
260	} ptr_list_struct;			
261	typedef struct {			
262	u_long error_count;			
263	ERROR error_el[1];			
264	} log_modeler_struct;			
265	typedef struct {			
266	u_long error_count;			
267	union {			
268	ERROR error_el[1];			
269	struct {			
270	u_long wear_count;			
271	u_long wear_id[1];			
272	char hostname[1];			
273	char username[1];			
274	} IULA_struct;			
275	} IULA_union;			
276	} log_wear_list_struct;			
277	typedef struct {			
278	u_long error_count;			
279	union {			
280	ERROR error_el[1];			
281	u_long value;			
282	} IUA_union;			
283	} log_uar_struct;			
284	typedef struct {			
285	u_long error_count;			
286	union {			
287	ERROR error_el[1];			
288	u_long value;			
289	} IIA_union;			
290	} log_lane_struct;			
291	typedef struct {			
292	u_long error_count;			
293	union {			
294	ERROR error_el[1];			
295	u_long value;			
296	} IPA_union;			
297	} log_pam_struct;			
298	typedef struct {			
299	u_long error_count;			
300	union {			
301	ERROR error_el[1];			
302	u_long value;			
303	} IPA_union;			
304	} log_pel_struct;			
305	typedef struct {			
306	u_long error_count;			
307	union {			
308	ERROR error_el[1];			
309	u_long device_count;			
310	u_long device_number[1];			
311	char device_name[1];			
312	} IDLA_struct;			
313	} IDLA_union;			
314	} log_device_list_struct;			
315	typedef struct {			
316	u_long error_count;			
317	union {			
318	ERROR error_el[1];			
319	char part_name[1];			
320	} IDWA_union;			
321	} log_device_name_struct;			
322	typedef struct {			
323	u_long error_count;			
324	union {			
325	ERROR error_el[1];			
326	u_long value;			
327	} IDA_union;			
328	} log_device_struct;			
329	typedef struct {			
330	u_long error_count;			
331	union {			
332	ERROR error_el[1];			
333	char string[1];			
334	} IDA_union;			
335	} log_dab_struct;			
336	typedef struct {			
337	u_long error_count;			
338	union {			
339	ERROR error_el[1];			
340	struct {			
341	u_long location_count;			
342	struct {			
343	u_long lane;			
344	u_long slot;			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/protans.h	DATE 5/23/89 TIME 1:20:23 pm	PAGE # 4/69
LINE #	HEADER TEXT			
361	} lase_slot[1],			
362	} IDLA_struct,			
363	} IDLA_union,			
364	} lsg_dab_loc_ana,			
365	typedef struct {			
366	u_long error_count,			
367	union {			
368	ERROR error_el[1],			
369	u_long value,			
370	} IIA_union,			
371	} lsg_instance_ana,			
372	typedef struct {			
373	u_long error_count,			
374	union {			
375	ERROR error_el[1],			
376	u_long value,			
377	} IFA_union,			
378	} lsg_fault_ana,			
379	typedef struct {			
380	u_long error_count,			
381	union {			
382	ERROR error_el[1],			
383	u_long value,			
384	} IFA_union,			
385	} lsg_avail_ptrn_ana,			
386	typedef struct {			
387	u_long error_count,			
388	ERROR error_el[1],			
389	} measurement_ana,			
390	typedef struct {			
391	u_long error_count,			
392	ERROR error_el[1],			
393	} extract_def_ana,			
394	typedef struct {			
395	u_long error_count,			
396	union {			
397	ERROR error_el[1],			
398	char definition_text[1],			
399	} EDA_union,			
400	} extract_def_ana,			
401	typedef struct {			
402	u_long error_count,			
403	ERROR error_el[1],			
404	} loop_ptrn_ana,			
405	typedef struct {			
406	u_long error_count,			
407	union {			
408	ERROR error_el[1],			
409	struct {			
410	u_short output_count,			
411	INIT_FIM_VALUE init_pis_value[1],			
412	} IIA_struct,			
413	} IIA_union,			
414	} reset_inst_ana,			
415	typedef struct {			
416	u_long total_number,			
417	u_long number[1],			
418	u_long checksum,			
419	} test_network_ana,			
420	typedef struct {			
421	u_long error_count,			
422	ERROR error_el[1],			
423	} abort_ana,			
424	typedef struct {			
425	u_long error_count,			
426	ERROR error_el[1],			
427	} begin_session_ana,			
428	typedef struct {			
429	u_long error_count,			
430	ERROR error_el[1],			
431	} label_dab_ana,			
432	typedef struct {			
433	u_long error_count,			
434	ERROR error_el[1],			
435	} eval_control_ana,			
436	typedef struct {			
437	u_long error_count,			
438	ERROR error_el[1],			
439	} reboot_ana,			
440	typedef struct {			
441	u_long error_count,			
442	ERROR error_el[1],			
443	} shutdown_ana,			
444	typedef struct {			
445	u_long error_count,			
446	ERROR error_el[1],			
447	} check_dabdef_ana,			
448	typedef struct {			
449	u_long error_count,			
450	ERROR error_el[1],			
451	} password_ana,			
452	/* Define a structure for version number and simulator type */			
453	#ifdef VMS			
454	typedef union {			
455	struct {			
456	unsigned sim_type:10;			
457	unsigned reserved:6;			
458	unsigned minor:8;			
459	unsigned major:8;			
460	} field;			
461	unsigned long version;			
462	} version_type;			
463	#else			
464	#endif PC386			

Copyright 1989 Logic Modeling Systems		HEADER FILE include/protans.h	DATE 5/23/89	PAGE # 5/70
LINE #	HEADER TEXT			
481				
482	typedef union {			
483	struct {			
484	unsigned sim_type:10;			
485	unsigned reserved:6;			
486	unsigned minor:8;			
487	unsigned major:8;			
488	} field;			
489	unsigned long version;			
490	} version_type;			
491				
492	} else			
493				
494	typedef union {			
495	struct {			
496	unsigned major:8;			
497	unsigned minor:8;			
498	unsigned reserved:6;			
499	unsigned sim_type:10;			
500	} field;			
501	unsigned long version;			
502	} version_type;			
503				
504	#endif			
505	#endif			
506				
507	/* Define a mask to extract sym_type */			
508	#define SIM_TYPE_MASK 0x3ff			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/protcmd.h	5/23/89	
			TIME	1/71
			1:20:23 pm	
LINE #	HEADER TEXT			
1	/* SCCS ID: protcmd.h rev 3.1.1, 4/24/89 at 07:34:40 */			
2				
3	/* BEGIN CONST DECL */			
4	#define DEAD_CMD 2			
5	#define WRITE_CMD 4			
6	#define CREATE_DEF_CMD 6			
7	#define CREATE_INSTANCE_CMD 8			
8	#define CREATE_FAULT_CMD 10			
9				
10	#define RELEASE_DEF_CMD 12			
11	#define RELEASE_INSTANCE_CMD 14			
12	#define RELEASE_FAULT_CMD 16			
13				
14	#define EVAL_CMD 18			
15				
16	#define SAVE_DEF_CMD 20			
17	#define SAVE_PTN_CMD 22			
18	#define SAVE_PTN_COUNT_CMD 24			
19	#define RESTORE_INST_CMD 26			
20	#define RESTORE_PTN_CMD 28			
21	#define PTN_MIST_CMD 30			
22				
23	#define INQ_MODELER_CMD 32			
24	#define INQ_USER_LIST_CMD 34			
25	#define INQ_USER_CMD 36			
26	#define INQ_LANE_CMD 38			
27	#define INQ_PAN_CMD 40			
28	#define INQ_PEL_CMD 42			
29	#define INQ_DEVICE_LIST_CMD 44			
30	#define INQ_DEVICE_NAME_CMD 46			
31	#define INQ_DEVICE_CMD 48			
32	#define INQ_DAS_CMD 50			
33	#define INQ_DAS_LOC_CMD 52			
34	#define INQ_INSTANCE_CMD 54			
35	#define INQ_FAULT_CMD 56			
36				
37	#define INQ_AVAIL_PTN_CMD 58			
38				
39	#define THEASUREMENT_CMD 60			
40				
41	#define EXTRACT_DEF_CMD 62			
42				
43	#define LOOP_PTN_CMD 64			
44				
45	#define RESET_INST_CMD 66			
46				
47	#define TEST_NETWORK_CMD 68			
48				
49	#define ABORT_CMD 70			
50				
51	#define BEGIN_SESSION_CMD 72			
52				
53	#define LABEL_DAS_CMD 74			
54				
55	#define EVAL_CONTROL_CMD 76			
56				
57	#define REBOOT_CMD 78			
58	#define SHUTDOWN_CMD 80			
59				
60	#define CHECK_DAMAGE_CMD 82			
61				
62	#define SAVE_DEF_COUNT_CMD 84			
63				
64	#define PASSWORD_CMD 86			
65				
66	#define NO_SUCH_CMD 88			
67				
68	/* END CONST DECL */			
69				
70				
71	typedef struct {			
72	u_short pin_number;			
73	u_long sin_time;			
74	u_char value;			
75	} INPUT_PIN_CHANGE;			
76				
77	typedef struct {			
78	u_long number;			
79	long units;			
80	char definition_text[11];			
81	} create_def_cmd;			
82				
83	typedef struct {			
84	u_short def_id;			
85	char device_info_string[11];			
86	} create_instance_cmd;			
87				
88	/* common ptrs count == -1 means that this fault should branch from whatever			
89	* pattern the associated instance currently has.			
90	* common ptrs count == positive number means that this fault should branch			
91	* from this pattern number of the associated instance. This is used in the			
92	* restore fault.			
93	*/			
94	typedef struct {			
95	u_short inst_id;			
96	long common_ptr_count;			
97	} create_fault_cmd;			
98				
99	typedef struct {			
100	u_short def_id;			
101	} release_def_cmd;			
102				
103	typedef struct {			
104	u_short inst_id;			
105	} release_instance_cmd;			
106				
107	typedef struct {			
108	u_short fault_id;			
109	u_short inst_id;			
110	} release_fault_cmd;			
111				
112	typedef struct {			
113	u_short inst_id;			
114	u_short data_pin_count;			
115	u_short eval_pin_count;			
116	u_short store_pin_count;			
117	INPUT_PIN_CHANGE data_pin_list[11];			
118	INPUT_PIN_CHANGE eval_pin_list[11];			
119	INPUT_PIN_CHANGE store_pin_list[11];			
120	} eval_cmd;			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/protcmd.h	5/23/89	
			TIME	2/72
			1:20:23 pm	
LINE #	HEADER TEXT			
121	typedef struct {			
122	u_short def_id,			
123	save_def_cmd,			
124	} save_def_cmd;			
125	typedef struct {			
126	u_long no_parameters_needed,			
127	save_def_cost_cmd,			
128	} save_def_cost_cmd;			
129	typedef struct {			
130	u_short inst_id,			
131	save_ptrn_cmd,			
132	} save_ptrn_cmd;			
133	typedef struct {			
134	u_short inst_id,			
135	save_ptrn_cost_cmd,			
136	} save_ptrn_cost_cmd;			
137	typedef struct {			
138	u_short inst_id,			
139	u_char check_input_count,			
140	u_char first_eval,			
141	u_char sample_count,			
142	u_long evaluation_count,			
143	u_char purge_on_next_eval,			
144	u_char has_history,			
145	u_char use_2_bit_per_pin,			
146	u_char enable_timing_meas,			
147	u_short pin_count,			
148	struct {			
149	u_short pin_number,			
150	u_char old_rsv,			
151	u_char old_filtered,			
152	u_char new_rsv,			
153	u_char new_filtered,			
154	u_long sim_time,			
155	} pin_info_array[1],			
156	} restore_inst_cmd;			
157	typedef struct {			
158	u_short inst_id,			
159	u_long unit_count,			
160	u_long pattern_count_follows,			
161	PRNG_RNGT_LONGWORD pattern,			
162	} restore_ptrn_cmd;			
163	typedef struct {			
164	u_short inst_id,			
165	u_char command,			
166	} ptrn_hist_cmd;			
167	typedef struct {			
168	u_long attribute,			
169	} inq_module_cmd;			
170	typedef struct {			
171	u_long no_parameters,			
172	inq_user_list_cmd,			
173	} inq_user_cmd;			
174	typedef struct {			
175	u_long user_number,			
176	} inq_user_cmd;			
177	typedef struct {			
178	u_long lane_number,			
179	} inq_lane_cmd;			
180	typedef struct {			
181	u_long lane_number,			
182	u_long pin_number,			
183	u_long attribute,			
184	} inq_pin_cmd;			
185	typedef struct {			
186	u_long lane_number,			
187	u_long slot_number,			
188	u_long attribute,			
189	} inq_slot_cmd;			
190	typedef struct {			
191	u_long no_parameters_needed,			
192	inq_device_list_cmd,			
193	} inq_device_list_cmd;			
194	typedef struct {			
195	u_long device number,			
196	inq_device_name_cmd,			
197	} inq_device_name_cmd;			
198	typedef struct {			
199	u_long device number,			
200	u_long attribute,			
201	} inq_device_cmd;			
202	typedef struct {			
203	u_long device number,			
204	u_long ddb_number,			
205	u_long attribute,			
206	} inq_dab_cmd;			
207	typedef struct {			
208	u_long device number,			
209	u_long ddb_number,			
210	u_long loc_number,			
211	} inq_dab_loc_cmd;			
212	typedef struct {			
213	u_short inst_id,			
214	u_long attribute,			
215	} inq_instance_cmd;			
216	typedef struct {			
217	u_short fault_id,			
218	u_long attribute,			
219	} inq_fault_cmd;			
220	typedef struct {			
221	u_short def_id,			
222	inq_avail_ptrn_cmd,			
223	} inq_avail_ptrn_cmd;			
224	typedef struct {			
225	u_short inst_id,			
226	u_char on_or_off,			
227	} measurement_cmd;			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/protcmd.h	5/23/89	3/73
			TIME	1:20:23 pm
LINE #	HEADER TEXT			
241				
242	typedef struct {			
243	u_short definition_id,			
244	} extract_def_cmd,			
245				
246	typedef struct {			
247	u_short inst_id,			
248	u_long loop_time,			
249	} loop_ptrn_cmd,			
250				
251	typedef struct {			
252	u_short inst_id,			
253	reset_inst_cmd,			
254				
255	typedef struct {			
256	u_long total_number,			
257	u_long number[1],			
258	u_long checksum,			
259	} test_network_cmd,			
260				
261	typedef struct {			
262	u_long no_parameter_needed,			
263	short_cmd,			
264				
265	typedef struct {			
266	u_char sim_type,			
267	char hostname[1],			
268	char username[1],			
269	} begin_session_cmd,			
270				
271	typedef struct {			
272	u_char lanes,			
273	u_char slots,			
274	u_char dab_type[1], /* NULL terminated string */			
275	u_char device_name[1], /* NULL terminated string */			
276	u_char manufacturer[1], /* NULL terminated string */			
277	u_char revision[1], /* NULL terminated string */			
278	u_char lanes_high,			
279	u_char slots_wide,			
280	u_char segment_number,			
281	} label_dab_cmd,			
282				
283	typedef struct {			
284	u_short inst_or_fault_id,			
285	u_long attributes,			
286	u_long value,			
287	} eval_control_cmd,			
288				
289	typedef struct {			
290	u_char flag,			
291	u_long seconds,			
292	} reboot_cmd,			
293				
294	typedef struct {			
295	u_char flag,			
296	u_long seconds,			
297	} shutdown_cmd,			
298				
299	typedef struct {			
300	char definition_text[1],			
301	} check_dabdef_cmd,			
302				
303	typedef struct {			
304	u_long attributes,			
305	char password[1],			
306	} password_cmd,			
307				

Copyright 1989 Logic Modeling Systems	HEADER FILE	DATE	PAGE #
	include/strng.h	5/23/89 1:20:23 pm	1/74

LINE #	HEADER TEXT
1	/* SOC9 ID: strng.h rev 1.1, 4/24/89 at 07:34:48 */
2	/*
3	Header file for String Module
4	*/
5	Types defined:
6	typedef unsigned short symbol;
7	/* symbol is an unsigned 16-bit number
8	Functions defined:
9	void strinit ();
10	/* initializes the string module data structures
11	void strfree ();
12	/* frees the string module data structures
13	symbol strtosym (char *str);
14	/* takes string, returns its symbol number
15	symbol strtol (char *str);
16	/* returns symbol number if already entered
17	symbol strtolst ();
18	/* returns last symbol number entered
19	char *symtostr (symbol sym);
20	/* takes symbol number, returns its string
21	void set_pin_name (symbol sym, char *name);
22	/* set the pin_name field of the passed symbol to the passed name
23	void set_pin_number (symbol sym, unsigned short num);
24	/* set the pin number field of the passed symbol to the passed number
25	void set_apin_name (symbol sym, char *name);
26	/* set the apin_name field of the passed symbol to the passed name
27	void set_apin_number (symbol sym, unsigned short num);
28	/* set the apin number field of the passed symbol to the passed number
29	void g_declare (symbol sym, tree T);
30	/* globally declare the passed symbol as the passed tree
31	void l_declare (symbol sym, tree T);
32	/* locally declare the passed symbol as the passed tree
33	unsigned short pin_name_of (symbol sym);
34	/* returns the pin_name field of the passed symbol
35	unsigned short pin_number_of (symbol sym);
36	/* returns the pin number field of the passed symbol
37	unsigned short apin_name_of (symbol sym);
38	/* returns the apin_name field of the passed symbol
39	unsigned short apin_number_of (symbol sym);
40	/* returns the apin number field of the passed symbol
41	tree g_lookup (symbol sym);
42	/* returns the global declaration for the passed symbol
43	tree l_lookup (symbol sym);
44	/* returns the local declaration for the passed symbol
45	tree lookup (symbol sym);
46	/* returns the (localist) declaration for the passed symbol
47	void strtstat ();
48	/* prints statistics about string module data structure
49	void strdump ();
50	/* dumps contents of string module data structure
51	*/
52	# define STATS 1 /* to include strtstat and strdump */
53	# define NULLSYM 0 /* no-null symbols are 1 or more */
54	typedef unsigned short symbol;
55	void
56	{
57	# ifdef STATS
58	strstat ();
59	/*strdump ();
60	# endif /* STATS */
61	strinit ();
62	strfree ();
63	set_pin_name ();
64	set_pin_number ();
65	set_apin_name ();
66	set_apin_number ();
67	g_declare ();
68	l_declare ();
69	set_xtra ();
70	symbol
71	strtosym ();
72	strtol ();
73	/*strtol ();
74	char *symtostr ();
75	unsigned short
76	pin_name_of ();
77	pin_number_of ();
78	apin_name_of ();
79	apin_number_of ();
80	g_lookup ();
81	l_lookup ();
82	/*lookup ();
83	set_xtra ();
84	/*
85	end of header file for String Module */
86	*/

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/task.h	5/23/89	1/75
			TIME	1:20:23 pm
LINE #	HEADER TEXT			
1	/* SOCS ID: task.h rev 3.1. 4/26/89 at 07:34:52 */			
2	/*			
3	** TASK.H			
4	** Task related info			
5	*/			
6	#define MOUTIMEOUT 0L			
7	/*			
8	** task info			
9	*/			
10	** Task ID's			
11	*/			
12	#define TRANSMIT_TASK_ID 40			
13	#define PATTERN_TASK_ID 50			
14	#define RECEIVE_TASK_ID 60			
15	#define NET_BOOT_TASK_ID 70			
16	#define SERIAL_TASK_ID 80			
17	#define KEYBOARD_TASK_ID 90			
18	#define HOUSEKEEPING_TASK_ID 100			
19	/*			
20	** Task Priority			
21	*/			
22	#define TRANSMIT_TASK_PRI 40			
23	#define PATTERN_TASK_PRI 50			
24	#define RECEIVE_TASK_PRI 60			
25	#define NET_BOOT_TASK_PRI 70			
26	#define SERIAL_TASK_PRI 80			
27	#define KEYBOARD_TASK_PRI 90			
28	#define HOUSEKEEPING_TASK_PRI 100			
29	/*			
30	#define DIAG_KEYBOARD_TASK_PRI 30			
31	#define DIAG_PATTERN_TASK_PRI 140			
32	#define DIAG_RECEIVE_TASK_PRI 50			
33	#define DIAG_SERIAL_TASK_PRI 60			
34	#define DIAG_TRANSMIT_TASK_PRI 80			
35	#define DIAG_HOUSEKEEPING_TASK_PRI 100			
36	/*			
37	** END OF TASK.H			
38	*/			

Copyright 1989 Logic Modeling Systems	HEADER FILE	DATE	PAGE #
	include/timer.h	5/23/89	1/76
	TIME	1:20:23 pm	

LINE #	HEADER TEXT
1	/* SOCS ID: timer.h rev 3.1, 4/24/89 at 07:34:55 */
2	/* Register offsets for 8254 on 128K CPU board */
3	#define TIMER_COUNTER0 0 /* Offset of timer counter 0 */
4	#define TIMER_COUNTER1 4 /* Offset of timer counter 1 */
5	#define TIMER_COUNTER2 8 /* Offset of timer counter 2 */
6	#define TIMER_CNTRL_WORD 0xc /* Offset of timer control word */
7	/* Constants for use in 8254 timer control word */
8	#define SELECT_TIMER_COUNTER0 0 /* control word, select timer 0 */
9	#define SELECT_TIMER_COUNTER1 1 /* control word, select timer 1 */
10	#define SELECT_TIMER_COUNTER2 2 /* control word, select timer 2 */
11	#define R_W_LB_TIMER_CNTRL 1 /* Read/write LB only */
12	#define R_W_HB_TIMER_CNTRL 2 /* Read/write HB only */
13	#define R_W_LB_HB_TIMER_CNTRL 3 /* Read/write LB and HB */
14	#define TIMER_MODE0 0 /* Interrupt on terminal count */
15	#define TIMER_MODE1 1 /* Hardware retriggerable one-shot */
16	#define TIMER_MODE2 2 /* Rate generator */
17	#define TIMER_MODE3 3 /* Square-wave mode */
18	#define TIMER_MODE4 4 /* Software-triggered strobe */
19	#define TIMER_MODE5 5 /* Hardware-triggered strobe */
20	#define BINARY 0 /* Count is binary */
21	#define BCD 1 /* Count is BCD */
22	/* Timer control word structure */
23	typedef struct
24	{
25	unsigned
26	select_timer_counter:2, /* SELECT_TIMER_COUNTER0/1/2 */
27	r_w_timer_cntrl:2, /* R_W (LB/HB) TIMER_CNTRL */
28	timer_mode:3, /* TIMER_MODE0/1/2/3/4/5 */
29	bcd:1, /* BINARY/BCD */
30	pad:24;
31	} timer_control;
32	/* Constants for use in 8254 timer read-back command */
33	#define READ_BACK_TIMER 3 /* Identifies read-back command */
34	#define READ_BACK_COUNT 1 /* ... of selected channel(s) */
35	#define READ_BACK_STATUS 2 /* ... of selected channel(s) */
36	#define READ_BACK_COUNT_AND_STATUS 0 /* ... of selected channel(s) */
37	#define READ_TIMER0 1 /* Identifies which channel(s) */
38	#define READ_TIMER1 2 /* to read-back. May be added */
39	#define READ_TIMER2 4 /* for multiple channels */
40	/* Read-back command structure */
41	typedef struct
42	{
43	unsigned
44	read_back:2, /* READ_BACK_TIMER */
45	count_status:2, /* READ_BACK_COUNT/STATUS */
46	counter:3, /* READ_TIMER0-2 */
47	zero:1, /* Must be 0 */
48	pad:24;
49	} timer_status;
50	/* Constants for use in parsing 8254 timer read-back status */
51	#define OUT_HIGH 1 /* Channel output is logic 1 */
52	#define OUT_LOW 0 /* Channel output is logic 0 */
53	#define NULL_COUNT 1 /* Counter hasn't loaded yet */
54	#define NOT_NULL_COUNT 0 /* Counter has loaded */
55	/* Read-back status structure */
56	typedef struct
57	{
58	unsigned
59	output:1, /* OUT_HIGH/OUT_LOW */
60	null_count:1, /* [NOT] NULL COUNT */
61	r_w_timer_cntrl:2, /* R_W (LB/HB) TIMER_CNTRL */
62	timer_mode:3, /* TIMER_MODE0/1/2/3/4/5 */
63	bcd:1, /* BINARY/BCD */
64	pad:24;
65	} counter_status;

Copyright 1989 Logic Modeling Systems	HEADER FILE include/tmg.h	DATE 5/23/89 TIME 1:20:24 pm	PAGE # 1/77
LINE #	HEADER TEXT		
1	/* SCCS ID: tmg.h rev 3.1, 4/21/89 at 07:34:58 */		
2	/*		
3	/* All registers are mapped to vary lane's address space.		
4	/* However, software should refer to the TMC only at 0x00000000.		
5	/*		
6	/* Except as noted, all bits in control registers 0 and 1 are set		
7	/* to 0 on power up.		
8	/*		
9	/* Differences between 1 and 2 bits per pin handling:		
10	/*		
11	/* cto = Intel 8254 timer chip		
12	/*		
13	/* Global lane defines used for R0:lane_enable and R3:lane_intr */		
14	#define LANE_A 0x0		
15	#define LANE_B 0x1		
16	#define LANE_C 0x2		
17	#define LANE_D 0x3		
18	typedef struct {		
19	/* Register 0 */		
20	unsigned :24,		
21	tmg_intr_clear:1, /* 0 clears timing gen. interrupt condition */		
22	backplane_error:1, /* 1 drives error on backplane, 0 doesn't */		
23	backplane_reset:1, /* 0 drives reset on backplane, 1 doesn't */		
24	tmg_reset:1, /* 0 resets timing gen., 1 enables */		
25	allow_clock_clear:1, /* 0 clears allow clock detector, 1 enables */		
26	ee_off_clear:1, /* 0 clears all edge/sample F/Fs, 1 enables */		
27	short_pattern_play:1, /* 1 aborts pattern play, auto-clearing */		
28	start_pattern_play:1, /* 1 starts pattern play, auto-clearing */		
29	/* Register 1 */		
30	unsigned :24,		
31	count:3, /* pattern clocks (divides 1) per logical clock */		
32	ctr_intr_clear:1, /* 0 clears output, a 0-1 starts the ctr. */		
33	ctr_intr_enable:1, /* 1 enables, 0 disables */		
34	tmg_intr_enable:1, /* 1 enables, 0 disables */		
35	lane_intr_enable:1, /* 1 enables lane error interrupts, 0 dis- */		
36	patterns_intr_enable:1, /* 1 enables pattern play interrupts, 0 dis- */		
37	/*		
38	/* Procedure for changing which clock is selected or changing the internal		
39	/* clock frequency (i.e., changing the value of R5:clock_select, or		
40	/* R3:pll_rate, or R3:pll_divisor).		
41	/*		
42	/* - deassert R5:clock_enable		
43	/* - verify that the clock is off by polling R5:clock_ee with timeout		
44	/* of 4 clocks @ 150KHz = 4*4.67us = 26.67us (should be implicit).		
45	/* - assert R5:clock_sync_clear		
46	/* - change R5:clock_select, or R3:pll_rate, or R3:pll_divisor		
47	/* - deassert R5:clock_sync_clear		
48	/* - assert R5:clock_enable		
49	/*		
50	/* Register 2: phased lock loop rate		
51	/*		
52	/* Allowed values are: 1-128 => 128 unique frequencies.		
53	/* Setting to 1 yields 100MHz, setting to 128 yields 300Hz.		
54	/* Software must load this register ASAP to allow for the most		
55	/* possible settling time (i.e., instant time to start pattern play).		
56	/*		
57	/* The equation for calculating the resultant pattern frequency is:		
58	/* f = 1 to 128 (R2,0-7)		
59	/* D = 2 to 256 (R3,0-7)		
60	/* S = 1, 2, 4 (R7,3-5)		
61	/*		
62	/* frequency = 60MHz * (256 / S)		
63	/*		
64	/* 256 * S * D		
65	/*		
66	/* Note that this is a general equation and can not be used across the		
67	/* entire frequency range. The desired pattern frequency must be divided		
68	/* into the following three groups with the indicated restrictions on the		
69	/* combinations of S and D:		
70	/*		
71	/* To get pattern clock rates between 100MHz and 300MHz:		
72	/* - choose source 0 from R7,3-5 (undivided phase loop output, S=1)		
73	/* - load R3,0-7 with k=256 to yield D=2 (to adjust the basic 60MHz		
74	/* phase locked loop output down to 100MHz)		
75	/* - adjust R=0-7 to get the desired frequency as specified above		
76	/*		
77	/* To get pattern clock rates between 30.6KHz and 150KHz:		
78	/* - choose source 1 from R7,3-5 (divide by 20 phase loop output, S=2)		
79	/* - adjust R=0-7 and D=R3,0-7 to get the desired frequency as		
80	/* specified above. FILL		
81	/*		
82	/* To get pattern clock rates between 19.3KHz and 55.6KHz:		
83	/* - choose source 2 from R7,3-5 (divide by 40 phase loop output, S=4)		
84	/* - load R3,0-7 with k=1 to yield D=256 (to adjust the basic 60MHz		
85	/* phase locked loop output down to 55.6KHz (=60MHz/4*256))		
86	/* - adjust R=0-7 to get the desired frequency as specified above		
87	/*		
88	unsigned :24,		
89	pll_rate:8, /* Initialized to 128 on power up */		
90	/* Register 3: phased lock loop divisor		
91	/*		
92	/* Software sets R3:pll_divisor to numbers k = 1 upto 255. This yields:		
93	/* divisor = D = 256 - (k - 1) = 256 down to 2		
94	/* Note that the minimum divisor, D, is 2, not 1.		
95	/*		
96	unsigned :24,		
97	pll_divisor:8, /* Initialized to 128 on power up. */		
98	/* NOTE: see above restriction on changing. */		
99	/* Register 4 */		
100	unsigned :24,		
101	sample_delay_range:2, /* nominal 0.5, 1, 2, 4ns delay ranges */		
102	edge_delay_range:2, /* nominal 0.5, 2, 10, 50ns delay ranges */		
103	lane_enable:4, /* choice of lanes to play patterns to, */		
104	/* see defines above. */		
105	/* R4:delay_range */		
106	#define RANGE_0_5 0 /* These defines are the nominal ranges that will */		
107	#define RANGE_2 1 /* vary by 1% to 2%. The actual ns/step is cal- */		
108	#define RANGE_10 2 /* culated at power up and used in all subsequent */		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/tmg.h	DATE 5/23/89	PAGE # 2/78
LINE #		HEADER TEXT		
121	121	/* calculations */		
122	122	#define RANGE_50 1		
123	123	/* Register 5 */		
124	124	unsigned :24,		
125	125	test_en,		
126	126	delay_clock_delay:6,		
127	127	/* Register 6 */		
128	128	/* 0: force sync clear. 1: don't force */		
129	129	/* 1: enables clock. 0: disables clock */		
130	130	/* single edge and single sample trigger */		
131	131	/* double edge and single sample trigger */		
132	132	/* sample trigger only */		
133	133	/* Register 7 */		
134	134	/* 0: force sync clear. 1: don't force */		
135	135	/* 1: enables clock. 0: disables clock */		
136	136	/* Select 1 of 5 clocks for pattern process */		
137	137	/* 0: 1: 2: 3: 4: 5: 6: 7: 8: 9: 10: 11: 12: 13: 14: 15: 16: 17: 18: 19: 20: 21: 22: 23: 24: 25: 26: 27: 28: 29: 30: 31: 32: 33: 34: 35: 36: 37: 38: 39: 40: 41: 42: 43: 44: 45: 46: 47: 48: 49: 50: 51: 52: 53: 54: 55: 56: 57: 58: 59: 60: 61: 62: 63: 64: 65: 66: 67: 68: 69: 70: 71: 72: 73: 74: 75: 76: 77: 78: 79: 80: 81: 82: 83: 84: 85: 86: 87: 88: 89: 90: 91: 92: 93: 94: 95: 96: 97: 98: 99: 100: 101: 102: 103: 104: 105: 106: 107: 108: 109: 110: 111: 112: 113: 114: 115: 116: 117: 118: 119: 120: 121: 122: 123: 124: 125: 126: 127: 128: 129: 130: 131: 132: 133: 134: 135: 136: 137: 138: 139: 140: 141: 142: 143: 144: 145: 146: 147: 148: 149: 150: 151: 152: 153: 154: 155: 156: 157: 158: 159: 160: 161: 162: 163: 164: 165: 166: 167: 168: 169: 170: 171: 172: 173: 174: 175: 176: 177: 178: 179: 180: 181: 182: 183: 184: 185: 186: 187: 188: 189: 190: 191: 192: 193: 194: 195: 196: 197: 198: 199: 200: 201: 202: 203: 204: 205: 206: 207: 208: 209: 210: 211: 212: 213: 214: 215: 216: 217: 218: 219: 220: 221: 222: 223: 224: 225: 226: 227: 228: 229: 230: 231: 232: 233: 234: 235: 236: 237: 238: 239: 240: 241: 242: 243: 244: 245: 246: 247: 248: 249: 250: 251: 252: 253: 254: 255: 256: 257: 258: 259: 260: 261: 262: 263: 264: 265: 266: 267: 268: 269: 270: 271: 272: 273: 274: 275: 276: 277: 278: 279: 280: 281: 282: 283: 284: 285: 286: 287: 288: 289: 290: 291: 292: 293: 294: 295: 296: 297: 298: 299: 300: 301: 302: 303: 304: 305: 306: 307: 308: 309: 310: 311: 312: 313: 314: 315: 316: 317: 318: 319: 320: 321: 322: 323: 324: 325: 326: 327: 328: 329: 330: 331: 332: 333: 334: 335: 336: 337: 338: 339: 340: 341: 342: 343: 344: 345: 346: 347: 348: 349: 350: 351: 352: 353: 354: 355: 356: 357: 358: 359: 360: 361: 362: 363: 364: 365: 366: 367: 368: 369: 370: 371: 372: 373: 374: 375: 376: 377: 378: 379: 380: 381: 382: 383: 384: 385: 386: 387: 388: 389: 390: 391: 392: 393: 394: 395: 396: 397: 398: 399: 400: 401: 402: 403: 404: 405: 406: 407: 408: 409: 410: 411: 412: 413: 414: 415: 416: 417: 418: 419: 420: 421: 422: 423: 424: 425: 426: 427: 428: 429: 430: 431: 432: 433: 434: 435: 436: 437: 438: 439: 440: 441: 442: 443: 444: 445: 446: 447: 448: 449: 450: 451: 452: 453: 454: 455: 456: 457: 458: 459: 460: 461: 462: 463: 464: 465: 466: 467: 468: 469: 470: 471: 472: 473: 474: 475: 476: 477: 478: 479: 480: 481: 482: 483: 484: 485: 486: 487: 488: 489: 490: 491: 492: 493: 494: 495: 496: 497: 498: 499: 500: 501: 502: 503: 504: 505: 506: 507: 508: 509: 510: 511: 512: 513: 514: 515: 516: 517: 518: 519: 520: 521: 522: 523: 524: 525: 526: 527: 528: 529: 530: 531: 532: 533: 534: 535: 536: 537: 538: 539: 540: 541: 542: 543: 544: 545: 546: 547: 548: 549: 550: 551: 552: 553: 554: 555: 556: 557: 558: 559: 560: 561: 562: 563: 564: 565: 566: 567: 568: 569: 570: 571: 572: 573: 574: 575: 576: 577: 578: 579: 580: 581: 582: 583: 584: 585: 586: 587: 588: 589: 590: 591: 592: 593: 594: 595: 596: 597: 598: 599: 600: 601: 602: 603: 604: 605: 606: 607: 608: 609: 610: 611: 612: 613: 614: 615: 616: 617: 618: 619: 620: 621: 622: 623: 624: 625: 626: 627: 628: 629: 630: 631: 632: 633: 634: 635: 636: 637: 638: 639: 640: 641: 642: 643: 644: 645: 646: 647: 648: 649: 650: 651: 652: 653: 654: 655: 656: 657: 658: 659: 660: 661: 662: 663: 664: 665: 666: 667: 668: 669: 670: 671: 672: 673: 674: 675: 676: 677: 678: 679: 680: 681: 682: 683: 684: 685: 686: 687: 688: 689: 690: 691: 692: 693: 694: 695: 696: 697: 698: 699: 700: 701: 702: 703: 704: 705: 706: 707: 708: 709: 710: 711: 712: 713: 714: 715: 716: 717: 718: 719: 720: 721: 722: 723: 724: 725: 726: 727: 728: 729: 730: 731: 732: 733: 734: 735: 736: 737: 738: 739: 740: 741: 742: 743: 744: 745: 746: 747: 748: 749: 750: 751: 752: 753: 754: 755: 756: 757: 758: 759: 760: 761: 762: 763: 764: 765: 766: 767: 768: 769: 770: 771: 772: 773: 774: 775: 776: 777: 778: 779: 780: 781: 782: 783: 784: 785: 786: 787: 788: 789: 790: 791: 792: 793: 794: 795: 796: 797: 798: 799: 800: 801: 802: 803: 804: 805: 806: 807: 808: 809: 810: 811: 812: 813: 814: 815: 816: 817: 818: 819: 820: 821: 822: 823: 824: 825: 826: 827: 828: 829: 830: 831: 832: 833: 834: 835: 836: 837: 838: 839: 840: 841: 842: 843: 844: 845: 846: 847: 848: 849: 850: 851: 852: 853: 854: 855: 856: 857: 858: 859: 860: 861: 862: 863: 864: 865: 866: 867: 868: 869: 870: 871: 872: 873: 874: 875: 876: 877: 878: 879: 880: 881: 882: 883: 884: 885: 886: 887: 888: 889: 890: 891: 892: 893: 894: 895: 896: 897: 898: 899: 900: 901: 902: 903: 904: 905: 906: 907: 908: 909: 910: 911: 912: 913: 914: 915: 916: 917: 918: 919: 920: 921: 922: 923: 924: 925: 926: 927: 928: 929: 930: 931: 932: 933: 934: 935: 936: 937: 938: 939: 940: 941: 942: 943: 944: 945: 946: 947: 948: 949: 950: 951: 952: 953: 954: 955: 956: 957: 958: 959: 960: 961: 962: 963: 964: 965: 966: 967: 968: 969: 970: 971: 972: 973: 974: 975: 976: 977: 978: 979: 980: 981: 982: 983: 984: 985: 986: 987: 988: 989: 990: 991: 992: 993: 994: 995: 996: 997: 998: 999: 1000		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/tmg.h	DATE 5/23/89	PAGE # 3/79
LINE #		HEADER TEXT		
241		/* Registers 32-63: IDPRON: read only */		
242		/* The IDPRON is a byte-wide resource on 32-bit boundaries */		
243		struct {		
244		unsigned :24, data:8,		
245		} id_prom(32);		
246		} TMC,		
247		/*		
248		* Use to map all the registers independent of bits during		
249		* some parts of diagnostics.		
250		*/		
251		typedef struct {		
252		struct {		
253		unsigned :24, data:8,		
254		} reg[64];		
255		} TMC_DIAG;		
256				
257				

Copyright 1989 Logic Modeling Systems		HEADER FILE include/tmg_def.h	DATE 5/23/89	PAGE # 1/80
LINE #		HEADER TEXT		
1	2	/* SOCR_ID: tmg_def.h rev 3.1. 4/24/89 at 07:35:01 */		
3	4	/*		
5	6	Definitions used in hardware diagnostics		
7	8	last modified 5/19/88 -- vkr		
9	10	added interrupt flag definitions 6/13/88 -- vkr		
11	12	changed to tmg_def.h, cleaned up in general		
13	14	*/		
15	16	/*		
17	18	Clock Board Address Definitions		
19	20	/*		
21	22	define CLOCK_BASE 0x00000000 /* base address		
23	24	define REG(1) ((u_char *)(((u_long)((CLOCK_BASE +		
25	26	define TMC_TIMER_OFFSET (24 * 4 + 3) /* byte offset of reg 0 */		
27	28	define TMC_INT !(((u_char *)0x00000003) & 0x04) /* TMC interrupt */		
29	30	/*		
31	32	Interrupt Flag Definitions		
33	34	/*		
35	36	define EOPLFLAG 1 /* End of Play Interrupt */		
37	38	define CYCFLAG 2 /* CYC Interrupt */		
39	40	define OSINTFLAG 4 /* On Board Interrupt */		
41	42	define BPERORFLAG 8 /* Backplane Error Interrupt */		
43	44	/*		
45	46	Miscellaneous */		
47	48	/*		
49	50	define in_tmg_lane_select(x) (tmgptr->lane_enable = (x))		
51	52	define EARLYTRAMPLETRIGGERMODE 0 /* fixed (early) sample trigger */		
53	54	define EDGE7SAMPLETRIGGERMODE 1 /* sample triggered by Edge 7 */		
55	56	define TOGGLEMODE 1		
57	58	define NOTOGGLEMODE 0		
59	60	define CHECKSUM_INIT 0x00		
61	62	define COUNT_MAX 70 /* 100 us */		
63	64	define TIMEOUT 1000 /* 1000 ms = 1 sec */		
65	66	define T_REF_IN_PS 8533333 /* reference period in ps */		
67	68	define RAMP_DWOP_TIME 15000 /* dump time in ps */		
69	70	define MAGIC_DEAD_TIME 9450 /* dead time in ps */		
71	72	/*		
73	74	7 ns for Magic Chip */		
75	76	/*		
77	78	1.7 ns for driver skew */		
79	80	/*		
81	82	0.75 ns for delay line skew */		
83	84	define TMC_THRESHES 10 /* number of thresholds to try */		
85	86	define TMC_CAL_TOGGLES 100 /* number of toggles used in cal */		
87	88	define START_DEAD_TIME_FUDGE 2 /* extra number of thresholds added		
89	90	to minimize threshold */		
91	92	define END_DEAD_TIME_FUDGE 2 /* same, but for max thresh		
93	94	/*		
95	96	CRC command definitions */		
97	98	/*		
99	100	define CTR0CEN 0x32 /* Counter 0 Control Word */		
101	102	define CTR1CEN 0x74 /* Counter 1 Control Word */		
103	104	define CTR2CEN 0x0e /* Counter 2 Control Word */		
105	106	define CTR3LATCH 0x0c /* latch count/status for 1,2,3 */		
107	108	define LATCH_CTR0_1 0x04		
109	110	define LATCH_CTR0_0 0x02		
111	112	/*		
113	114	Backplane macros */		
115	116	/*		
117	118	define PLAY_MODE 1		
119	120	define ACCESS_MODE 0		
121	122	/*		
123	124	define Lane_select(x) (tmgptr->lane_enable = (x))		
125	126	define Get_Bp_error() (tmgptr->lane_intr)		
127	128	define Bp_reset_on() (tmgptr->backplane_resetL = 0)		
129	130	define Bp_reset_off() (tmgptr->backplane_resetL = 1)		
131	132	define Bp_mode() (tmgptr->backplane_mode ? PLAY_MODE : ACCESS_MODE)		
133	134	/*		
135	136	Calibration definitions */		
137	138	extern void tmg_adjnst_delay();		
139	140	/*		
141	142	ifdef OLD_DELAY_CRT		
143	144	define tmg_as_to_dly_reg(x) (((x)<27)? \		
145	146	(((x)>27)? \		
147	148	(((x)>31)? \		
149	150	(((x)>31)? \		
151	152	define tmg_dly_reg_to_as(x) (((x) & 0x30) >> 1) + ((x) & 0x07)		
153	154	/*		
155	156	define tmg_as_to_dly_reg(x) (((x)>27)? \		
157	158	define tmg_dly_reg_to_as(x) (((x)>27)? \		
159	160	/*		
161	162	define DETECT_MODE 0		
163	164	define DETECT_COUNT 1		
165	166	define DETECT_LEN 1		
167	168	define DETECT_SAMPLE 4		
169	170	/*		
171	172	define tmg_set_slot_count(n) (tmgptr->slot_count = (n) - 1)		
173	174	/*		
175	176	Edge ramp definitions */		
177	178	/*		
179	180	define tmg_get_armp() (tmgptr->edge_delay_range)		
181	182	define tmg_set_armp(ramp) (tmgptr->edge_delay_range = (ramp))		
183	184	/*		
185	186	define tmg_set_threshold(edge, threshold) \		
187	188	{ tmgptr->edge_delay[edge].delay = (threshold); \		
189	190	{ int count = 1001; while(--count); }		
191	192	/*		
193	194	define tmg_get_edge_cal() (tmgptr->edge_cal)		
195	196	/*		
197	198	define tmg_get_test_mode() (tmgptr->test_mode)		
199	200	define tmg_set_test_mode(x) (tmgptr->test_mode = (x))		
201	202	define tmg_get_delay_reg() (tmgptr->delay_clock_delay)		
203	204	define tmg_set_delay_reg(x) (tmgptr->delay_clock_delay = (x))		
205	206	define tmg_get_delay() tmg_dly_reg_to_as(tmg_get_delay_reg())		
207	208	define tmg_set_delay(x) tmg_set_delay_reg(tmg_as_to_dly_reg(x))		
209	210	/*		
211	212	Sample ramp definitions */		
213	214	/*		
215	216	define tmg_set_edge_7_threshold(threshold) \		
217	218	{ tmgptr->sample_trigger_threshold = (threshold); \		
219	220	{ int count = 1001; while(--count); }		
221	222	/*		
223	224	define tmg_get_armp() (tmgptr->sample_delay_range)		
225	226	define tmg_set_armp(armp) (tmgptr->sample_delay_range = (armp))		
227	228	/*		
229	230	define tmg_set_sample_threshold(threshold) \		

Copyright 1989 Logic Modeling Systems		HEADER FILE include/tmg_def.h	DATE 5/23/89 TIME 1:20:24 pm	PAGE # 2/81
LINE #	HEADER TEXT			
121	{tmgptr->sample_delay = (threshold)); \			
122	{ 1st count = 1001, while(--count); }			
123				
124	#define tmg_get_sample_val() (tmgptr->sample_val)			
125	#define tmg_get_sample_width_reg(width) (tmgptr->sample_width = width)			
126	#define tmg_get_sample_width_reg() (tmgptr->sample_width)			
127				
128	#define TMG_MIN_SAMPLE_WIDTH 50000 /*to prevent pac error during calib*/			
129				
130	/*			
131	tmg_predict_period and tmg_predict_threshold assume the following units:			
132	/*			
133	period piconseconds			
134	threshold millivolts			
135	slope millivolts/millisecons			
136	offset piconseconds			
137	*/			
138				
139	#ifndef PRECISION			
140	#define tmg_predict_period(threshold, slope, offset) \			
141	(((long)(offset) + ((long)(threshold))*((long)(slope))))			
142				
143	#define tmg_predict_threshold(period, slope, offset) \			
144	(((long)(period) - (long)(offset)) / (long)(slope))			
145				
146	#define tmg_predict_offset(period, threshold, slope) \			
147	(((long)(period) - ((long)(threshold))*((long)(slope))))			
148	#else PRECISION			
149	#define tmg_predict_period(threshold, slope, offset) \			
150	(((long)((double)(offset) \			
151	+ ((double)(threshold))*((double)(slope))))			
152				
153	#define tmg_predict_threshold(period, slope, offset) \			
154	(((long)((double)(period - (offset))) / (double)(slope)))			
155				
156	#define tmg_predict_offset(period, threshold, slope) \			
157	(((long)((double)(period) \			
158	- ((double)(threshold))*((double)(slope))))			
159	#endif PRECISION			
160				
161	#define NUMBER_OF_FRAMES 4			
162	#define NUMBER_OF_EDGES 4			
163	#define NUMBER_OF_SAMPLES 4			
164				
165	#define EDGE_0 0			
166	#define EDGE_1 1			
167	#define EDGE_2 2			
168	#define EDGE_3 3			
169	#define EDGE_4 4			
170	#define EDGE_5 5			
171	#define SAMPLE_EDGE 6			
172				
173	/*			
174	A global pointer to the following structure is used by			
175	tmg_calibrator() who fills in the values. The values			
176	are defined as follows:			
177	/*			
178	EdgeMinThresh: Array of minimum threshold settings			
179	for each of 4 ranges for each of 4			
180	edges below which edges cannot be			
181	(reliably) generated.			
182	/*			
183	EdgeMinDelay: Array of settings for each of 4 ranges			
184	below which edge delays may not be			
185	specified.			
186	/*			
187	EdgeMaxDelay: Array of settings for each of 4 ranges			
188	that represent the maximum allowable			
189	delay setting.			
190	Setting a delay above this value may			
191	result in no edge.			
192	/*			
193	EdgeOffset: Array of offsets for each of 4 ranges			
194	for each of 4 edges, together with the			
195	slope, describe completely the relationship			
196	of the edge delay with the threshold setting.			
197	/*			
198	EdgeSlope: Array of ratios for each of 4 ranges.			
199	Average of 4 edges that represent			
200	piconseconds per bit of delay setting.			
201	/*			
202	EdgeLinearity: Array of linearity measurements for the slope			
203	of the range. 0 = perfect linearity.			
204	/*			
205	Edge7MinThresh: Array of settings for each of 4 ranges			
206	that represent the minimum allowable			
207	threshold setting for Edge[7]. Below these			
208	values an edge (and Sample Trigger) may not			
209	be reliably generated.			
210	/*			
211	SampleMinThresh: Array of settings for each of 4 ranges			
212	that represent the minimum allowable			
213	threshold setting for the sample ramp. Below			
214	these values, the sample may not be reliably			
215	generated.			
216	/*			
217	SampleMinDelay: Array of minimum delay settings for			
218	sample - each of 4 ranges.			
219	/*			
220	SampleMaxDelay: Array of maximum delay settings for			
221	sample for each of 4 ranges. Setting			
222	a delay above this value may result			
223	in no sample.			
224	/*			
225	SampleSlope: Array of ratios for each of 4 ranges			
226	that represent piconseconds per bit of			
227	sample delay setting.			
228	/*			
229	SampleOffset: Array of offsets for each of 4 Edge			
230	ranges for each of 4 sample ranges.			
231	The numbers represent nanoseconds of			
232	delay from pattern clock when both			
233	Edge[7] is set at Edge7MinDelay and Sample			
234	is set at 0.			
235	/*			
236	SampleLinearity: Array of linearity measurements for the slope			
237	of the range. 0 = perfect linearity.			
238	/*			
239	EarlySampleMinDelay: Array of minimum delays available with the			
240	Early Sample Trigger Mode.			

Copyright 1989 Logic Modeling Systems		HEADER FILE	DATE	PAGE #
		include/trng_def.h	TIME	3/82
LINE #	HEADER TEXT			
241	EarlySampleMaxDelay: Array of maximum delays available with the			
242	Early Sample Trigger Mode.			
243	EarlySampleOffset: Array of offsets			
244				
245				
246	DelayDelay: This is the value used for the delay clock			
247	delay line setting.			
248				
249	DumpDelay: This is the value used for the dump delay			
250	line setting.			
251				
252				
253	CalCompleted: This is a flag (TRUE or FALSE) that indicates			
254	to whoever cares whether the calibration has			
255	been performed or not.			
256				
257	*/			
258				
259	struct CALIB {			
260	u_long EdgeMinThresh(NUMBER_OF_EDGES){NUMBER_OF_EDGES},			
261	u_long EdgeMinDelay(NUMBER_OF_EDGES),			
262	u_long EdgeMaxDelay(NUMBER_OF_EDGES),			
263	long EdgeOffset(NUMBER_OF_EDGES){NUMBER_OF_EDGES},			
264	u_long EdgeSlope(NUMBER_OF_EDGES){NUMBER_OF_EDGES},			
265	u_long EdgeLinearity(NUMBER_OF_EDGES){NUMBER_OF_EDGES},			
266				
267	u_long Edge7MinThresh(NUMBER_OF_EDGES),			
268	u_long Edge7Slope(NUMBER_OF_EDGES),			
269	u_long Edge7Linearity(NUMBER_OF_EDGES),			
270	u_long SampleMinThresh(NUMBER_OF_EDGES),			
271	u_long SampleMinDelay(NUMBER_OF_EDGES),			
272	u_long SampleMaxDelay(NUMBER_OF_EDGES),			
273	u_long SampleSlope(NUMBER_OF_EDGES),			
274	u_long SampleLinearity(NUMBER_OF_EDGES),			
275	long SampleOffset(NUMBER_OF_EDGES){NUMBER_OF_EDGES},			
276	long EarlySampleMinDelay(NUMBER_OF_EDGES),			
277	u_long EarlySampleMaxDelay(NUMBER_OF_EDGES),			
278	long EarlySampleOffset(NUMBER_OF_EDGES),			
279	u_long DelayDelay,			
280	u_long DumpDelay,			
281	u_long CalCompleted,			
282	};			

Copyright 1989 Logic Modeling Systems	HEADER FILE include/trees.h	DATE 5/23/89 TIME 1:20:24 pm	PAGE # 1/83
LINE #	HEADER TEXT		
1	/* SCCS ID: trees.h rev. 1.1, 4/24/89 at 07:35:04 */		
2	/*		
3	* header file for tree module		
4	* types needed:		
5	* type symbol from strings.h		
6	* trees defined		
7	* typedef unsigned short tree		
8	* functions defined:		
9	* void treeinit();		
10	* initializes the tree data structure		
11	* void treefree();		
12	* frees the tree data structure		
13	* void hldtrm (name, text); symbol name, text		
14	* build terminal node with passed name, text		
15	* void hldstrm (name, kids); symbol name, unsigned short kids;		
16	* build nonterminal node with passed name over kids		
17	* tree poptr ();		
18	* pop and returns completely built tree		
19	* symbol nameofnode (T); tree T		
20	* returns name of root node of passed tree		
21	* symbol textofnode (T); tree T		
22	* returns text of root node of passed tree if node is terminal		
23	* void set_text (T, text); tree T, symbol text;		
24	* sets the text of the passed node to the passed symbol		
25	* unsigned short kidcount (T); tree T		
26	* returns number of subtrees of root node of passed tree		
27	* tree subtree (T, kid); unsigned short kid; tree T;		
28	* returns the kidth subtree of the passed tree		
29	* unsigned short whichkid (T, kid); tree T, kid;		
30	* returns a where kid is the kidth subtree of T		
31	* symbol file of (T); tree T;		
32	* returns the source file for this node		
33	* symbol line of (T); tree T;		
34	* returns the source line for this node		
35	* void decorates (T, dec); tree T, long dec;		
36	* decorates the root node of the passed tree with the passed decoration		
37	* long dec_ration (T); tree T;		
38	* returns the decoration on the root node of the passed tree		
39	* void printtree (T); tree T;		
40	* prints the passed tree		
41	* void printnode (T); tree T;		
42	* prints the root node of the passed tree		
43	*/		
44	# define DECOR /* to include decorates and dec_ration */		
45	# define NULLTREE 0 /* the null tree */		
46	typedef unsigned short tree;		
47	struct node /* node structure */		
48	{ symbol name, /* name of node */		
49	unsigned short kids, /* kid count */		
50	unsigned short first, /* index of first kid or text of node if terminal */		
51	/* ASSUMES: a symbol is an unsigned short (should use a union) */		
52	# ifdef DECOR		
53	long dec; /* decoration */		
54	# endif /* DECOR */		
55	symbol arc, /* the source file name */		
56	unsigned short arcline, /* the source line number */		
57	};		
58	void		
59	treeinit ();		
60	treefree ();		
61	hldtrm ();		
62	hldstrm ();		
63	set_text ();		
64	# ifdef DECOR		
65	decorates ();		
66	# endif /* DECOR */		
67	printtree ();		
68	printnode ();		
69	tree		
70	poptr ();		
71	subtree ();		
72	symbol		
73	nameofnode ();		
74	textofnode ();		
75	file of ();		
76	unsigned short		
77	kidcount ();		
78	/* whichkid */		
79	line of ();		
80	# ifdef DECOR		
81	long dec_ration ();		
82	# endif /* DECOR */		
83	/* end of header file for tree module */		
84	/		

Copyright 1989 Legit Modeling Systems		HEADER FILE	DATE	PAGE #
		include/vrtx.h	5/23/89	1/84
			TIME	1:20:24 pm
LINE #	HEADER TEXT			
1	// SOCH ID: vrtx.h rev 3.1 4/24/89 at 07:35:07 //			
2	/*			
3	** When making use of the lm_delay() call,			
4	** use of constants will be rather utilization of C++ time.			
5	** NOTE: The system timer tick is 5 ms.			
6	** lm_delay() expects number of milliseconds to delay.			
7	** It will delay a minimum of 5 ms, typical min delay = 7.5 ms.			
8	** lm_time() returns number of milliseconds since power up.			
9	** (rounded to the 5ms system timer tick).			
10	** lm_tick is a long integer by which user to calculate			
11	** number of ticks since beginning of time.			
12	** lm_number_of_ticks()			
13	** Given number of milliseconds calculate the number			
14	** of system ticks.			
15	*/			
16	extern unsigned long lm_tick;			
17	#define lm_number_of_ticks(x) (((x - 5) / 5) + 2)			
18	#define lm_delay(x) (sc_delay((x - 5) / 5) + 2)			
19	#define lm_time() ((lm_tick < 2) + lm_tick)			
20	#define VRTX_OK 0x0			
21	#define VRTX_TIMEOUT 0x1			

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM	DATE	PAGE #
	sfi/access.c	5/23/89	1/1
		TIME	1:20:50 pm

LINE #	SOURCE TEXT
1	/* SCCS_ID: access.c rev 3.2, 5/9/87 at 16:34:28 */
2	
3	#include <stdio.h>
4	#include "common.h"
5	#include "network.h"
6	#include "lm_sfi.h"
7	#include "lsfi.h"
8	#include "lm_rd_wr.h"
9	#include "protcmd.h"
10	#include "protans.h"
11	#include "message.h"
12	#include "globals.h"
13	
14	/* If you change this message you have to make the corresponding change
15	in bootprom.c.
16	*/
17	static char waiting_to_be_booted_msg[] = "Modeler waiting to be booted";
18	static char not_properly_installed_msg[] = "Modeler is not properly installed";
19	static char old_eproms_msg[] = "Must install new CPU EPROMS; call field engineering";
20	static char old_boots_msg[] = "Must use newer version of utilities to install modeler";
21	
22	lm_read(modeler_name,
23	memory_type, ee_number, offset, number_of_bytes, buffer, status)
24	char
25	memory_type,
26	u_long ee_number,
27	u_long offset,
28	register u_long number_of_bytes,
29	register char *buffer;
30	u_long *status;
31	{
32	
33	/* read from modeler memory space
34	*/
35	long ret;
36	short i;
37	
38	#ifdef DEBUG
39	if (lm_debug[1]) {
40	PRINTF(("lm_read(%s, %d, %d, %d, %d, %08x, %08x)\n",
41	modeler_name, memory_type, ee_number, offset, number_of_bytes,
42	buffer, status));
43	}
44	#endif
45	
46	if (lm_mopen(modeler_name) == LM_ERROR) return LM_ERROR;
47	
48	/* Do sanity check */
49	if (number_of_bytes > MAX_RD_WR_BUFFER_SIZE) {
50	lm_queue_message(ERROR_MSG,
51	"internal error: maximum buffer size exceeded");
52	return(LM_ERROR);
53	}
54	
55	lm_put_int(READ_CMD);
56	
57	/* memory type to read from */
58	lm_put_int(memory_type);
59	
60	/* ee number */
61	lm_put_int(ee_number);
62	
63	/* offset */
64	lm_put_int(offset);
65	
66	/* number of bytes */
67	lm_put_int(number_of_bytes);
68	
69	if (! lm_end_put(lm_modeler_fd))
70	return(LM_ERROR);
71	
72	if (lm_get_int() != READ_MSG) {
73	lm_queue_message(ERROR_MSG,
74	"internal error: wrong reply received from modeler: %s",
75	modeler_name);
76	return(LM_ERROR);
77	}
78	
79	if ((ret = lm_dequeue_errors()) == LM_ERROR)
80	return(LM_ERROR);
81	
82	/* copy number of bytes to buffer */
83	for (i=0; i < number_of_bytes; ++i)
84	*buffer++ = lm_get_char();
85	
86	return(ret);
87	}
88	
89	lm_write(modeler_name,
90	memory_type, ee_number, offset, number_of_bytes, buffer, status)
91	char
92	memory_type,
93	u_long ee_number,
94	u_long offset,
95	register u_long number_of_bytes,
96	register char *buffer;
97	u_long *status;
98	{
99	/*
100	write to modeler memory space
101	*/
102	#ifdef DEBUG
103	if (lm_debug[1]) {
104	PRINTF(("lm_write(%s, %d, %d, %d, %d, %08x, %08x)\n",
105	modeler_name, memory_type, ee_number, offset, number_of_bytes,
106	buffer, status));
107	}
108	#endif
109	
110	if (lm_mopen(modeler_name) == LM_ERROR) return LM_ERROR;
111	return lm_write(memory_type, ee_number, offset,
112	number_of_bytes, buffer, status);
113	}
114	
115	lm_mopen(modeler_name)
116	char
117	modeler_name,
118	{
119	clear_errors();
120	if (!lm_sim_verified) {

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/access.c	DATE 5/23/89	PAGE # 2/2
LINE #		SOURCE TEXT		
121		lm_queue_message(ERROR_MSG,		
122		"internal simulator error: lm_begin_modeling_session() not yet been called",		
123		return(LM_ERROR);		
124		}		
125		if (check_connection(modeler_name) == FAILURE)		
126		return(LM_ERROR);		
127		lm_choose_conn(lm_modeler_fd);		
128		return lm_dequeue_errors();		
129		}		
130		lm_write(memory_type, ee_number, offset, number_of_bytes, buffer, status)		
131		{		
132		register u_long memory_type;		
133		register u_long ee_number;		
134		register u_long offset;		
135		register u_long number_of_bytes;		
136		register char *buffer;		
137		register u_long *status;		
138		{		
139		register u_long i;		
140		{		
141		status = 0;		
142		/* Do a sanity check... */		
143		if (number_of_bytes > MAX_RD_WR_BUFFER_SIZE) {		
144		lm_queue_message(ERROR_MSG,		
145		"internal error: maximum buffer size exceeded",		
146		return(LM_ERROR);		
147		}		
148		lm_put_int(WRITE_CMD);		
149		/* memory_type to write to */		
150		lm_put_int(memory_type);		
151		/* ee_number */		
152		lm_put_int(ee_number);		
153		/* offset */		
154		lm_put_int(offset);		
155		/* number_of_bytes */		
156		lm_put_int(number_of_bytes);		
157		/* copy the data from the buffer to outgoing buffer... */		
158		for (i = 0; i < number_of_bytes; ++i)		
159		lm_put_char(*buffer++);		
160		if (!lm_end_put(lm_modeler_fd))		
161		return(LM_ERROR);		
162		if (lm_get_int() != WRITE_ACK) {		
163		lm_queue_message(ERROR_MSG,		
164		"internal error: wrong reply received from modeler: %s",		
165		lm_modeler_name_selected);		
166		return(LM_ERROR);		
167		}		
168		return lm_dequeue_errors();		
169		}		
170		}		
171		static		
172		check_connection(modeler_name)		
173		{		
174		char *modeler_name;		
175		{		
176		short akt;		
177		u_char i;		
178		u_char removed_error;		
179		{		
180		if (lm_modeler_fd != -1) {		
181		if (strcmp(lm_modeler_name_selected, modeler_name) == 0)		
182		return(SUCCESS);		
183		else {		
184		/* The user is trying to talk to different modeler;		
185		* close the connection to the previous modeler.		
186		*/		
187		for (i = 0; lm_system_table[i] != NULL; ++i) {		
188		if (strcmp(lm_system_table[i]->name, lm_modeler_name_selected) == 0) {		
189		break;		
190		}		
191		if (lm_system_table[i] == NULL) {		
192		lm_queue_message(ERROR_MSG, "internal error: previously selected modeler: %s is missing from the system table",		
193		lm_modeler_name_selected);		
194		return(FAILURE);		
195		}		
196		if (lm_system_table[i]->state == USED_FOR_RD_WR) {		
197		lm_choose_conn(lm_modeler_fd);		
198		lm_put_int(ABORT_CMD);		
199		(void)lm_end_put(lm_modeler_fd);		
200		close_modeler(lm_modeler_fd);		
201		lm_system_table[i]->state = NOT_OPENED;		
202		lm_system_table[i]->fd = -1;		
203		lm_modeler_fd = -1;		
204		}		
205		for (i = 0; lm_system_table[i] != NULL; ++i) {		
206		if (strcmp(lm_system_table[i]->name, modeler_name) == 0) {		
207		break;		
208		}		
209		if (lm_system_table[i] == NULL) {		
210		lm_queue_message(WARNING_MSG, "modeler: %s is not in %s file",		
211		modeler_name, HOSTFILENAME);		
212		/* allow user to specify modeler not in HOSTFILENAME */		
213		if (lm_enter_system(modeler_name, -1) == FAILURE)		
214		return(FAILURE);		
215		}		
216		switch (lm_system_table[i]->state) {		
217		case USED:		
218		break;		
219		}		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/access.c

DATE 5/23/89
TIME 1:20:50 pm

PAGE #
3/3

```

LINE # SOURCE TEXT
241 case USED_FOR_RD_WR:
242 break;
243 case UNAVAIL:
244 lm_queue_message(ERROR_MSG, "modeler: (L) unavailable",
245 modeler_name);
246 return(FAILURE);
247 case NOT_OPENED:
248 if (lm_initconn(modeler_name, lakt) == FAILURE) {
249 /*lm_system_table[i]->state = UNAVAIL;*/
250 return(FAILURE);
251 }
252 else {
253 if (lm_send_begin_session_cmd((char)lakt, modeler_name) == FAILURE) {
254 removed_error = FALSE;
255 if (lm_remove_message(ERROR_MSG,
256 old_hosts_msg) == SUCCESS)
257 removed_error = TRUE;
258 if (lm_remove_message(ERROR_MSG,
259 old_proms_msg) == SUCCESS)
260 removed_error = TRUE;
261 if (lm_remove_message(ERROR_MSG,
262 waiting_to_be_booted_msg) == SUCCESS)
263 removed_error = TRUE;
264 if (lm_remove_message(ERROR_MSG,
265 not_properly_installed_msg) == SUCCESS)
266 removed_error = TRUE;
267 if (removed_error == FALSE) {
268 lm_close_modeler(lakt);
269 return(FAILURE);
270 }
271 else {
272 lm_system_table[i]->state = USED_FOR_RD_WR;
273 lm_system_table[i]->fd = lakt;
274 }
275 }
276 else {
277 lm_system_table[i]->state = USED;
278 lm_system_table[i]->fd = lakt;
279 }
280 break;
281 default:
282 lm_queue_message(ERROR_MSG, "internal error: unknown modeler accessibility state");
283 return(FAILURE);
284 }
285 lm_modeler_fd = lm_system_table[i]->fd;
286 (void)strcpy(lm_modeler_name_selected, modeler_name);
287 return(SUCCESS);
288 }
289 lm_close_read_write_connection()
290 {
291 u_char i;
292 for (i = 0; lm_system_table[i] != NULL; ++i) {
293 if (lm_system_table[i]->state == USED_FOR_RD_WR) {
294 lm_choose_conn(lm_system_table[i]->fd);
295 lm_put_int(ABORT_CMD);
296 (void)lm_end_put(lm_system_table[i]->fd);
297 lm_close_modeler(lm_system_table[i]->fd);
298 lm_system_table[i]->state = NOT_OPENED;
299 lm_system_table[i]->fd = -1;
300 }
301 }
302 lm_modeler_fd = -1;
303 return(SUCCESS);
304 }
305 lm_close_connection_after_boot(modeler_name)
306 char *modeler_name;
307 {
308 u_char i;
309 for (i = 0; lm_system_table[i] != NULL; ++i) {
310 if (strcmp(lm_system_table[i]->name, modeler_name) == 0) {
311 break;
312 }
313 }
314 if (lm_system_table[i] != NULL) {
315 if (lm_system_table[i]->state == USED_FOR_RD_WR) {
316 lm_close_modeler(lm_system_table[i]->fd);
317 lm_system_table[i]->state = NOT_OPENED;
318 }
319 }
320 lm_modeler_fd = -1;
321 return(SUCCESS);
322 }

```

Copyright 1989 Logic Modeling Systems		HEADER FILE sfi/conv.h	DATE 5/23/89 TIME 1:20:51 pm	PAGE # 1/4
HEADER TEXT				
LINE #				
1	/* SCCS ID: conv.h rev 1.1. 4/24/89 at 07:35:35 */			
2	/* CONVERSION MACROS */			
3	typedef struct {			
4	union {			
5	#ifdef VAX			
6	struct {u_char lcc_b0, lcc_b1, lcc_b2, lcc_b3, } lcc_us_b;			
7	struct {u_short lcc_w0, lcc_w1, } lcc_us_w;			
8	#else			
9	struct {u_char lcc_b3, lcc_b2, lcc_b1, lcc_b0, } lcc_us_b;			
10	struct {u_short lcc_w1, lcc_w0, } lcc_us_w;			
11	#endif			
12	u_long lcc_long;			
13	} lcc_us;			
14	#define lcc_long lcc_us.lcc_long			
15	#define lcc_short lcc_us.lcc_us_w.lcc_w0			
16	#define lcc_char lcc_us.lcc_us_b.lcc_b0			
17	#define lcc_b3 lcc_us.lcc_us_b.lcc_b3			
18	#define lcc_b2 lcc_us.lcc_us_b.lcc_b2			
19	#define lcc_b1 lcc_us.lcc_us_b.lcc_b1			
20	#define lcc_b0 lcc_us.lcc_us_b.lcc_b0			
21	#define lcc_w1 lcc_us.lcc_us_w.lcc_w1			
22	#define lcc_w0 lcc_us.lcc_us_w.lcc_w0			
23	#define lcc_w0 lcc_us.lcc_us_w.lcc_w0			
24	#define lcc_w0 lcc_us.lcc_us_w.lcc_w0			
25	} CONVERSION_RECORD;			
26	#define write_long(fn, uval) { CONVERSION_RECORD xval; \			
27	xval.lcc_long = (long)uval; \			
28	(void)putc(xval.lcc_b3, fn); \			
29	(void)putc(xval.lcc_b2, fn); \			
30	(void)putc(xval.lcc_b1, fn); \			
31	(void)putc(xval.lcc_b0, fn); \			
32	if (ferror(fn)) { \			
33	lm_queue_message(ERROR_MSG, "error writing save-set, check quota and file system"); \			
34	return(LM_ERROR); \			
35	} \			
36	#define write_short(fn, uval) { CONVERSION_RECORD xval; \			
37	xval.lcc_short = (short)uval; \			
38	(void)putc(xval.lcc_b1, fn); \			
39	(void)putc(xval.lcc_b0, fn); \			
40	if (ferror(fn)) { \			
41	lm_queue_message(ERROR_MSG, "error writing save-set, check quota and file system"); \			
42	return(LM_ERROR); \			
43	} \			
44	#define write_char(fn, uval) { (void)putc((char)uval, fn); \			
45	if (ferror(fn)) { \			
46	lm_queue_message(ERROR_MSG, "error writing save-set, check quota and file system"); \			
47	return(LM_ERROR); \			
48	} \			
49	#define read_long(fn, uval) { CONVERSION_RECORD xval; \			
50	xval.lcc_b3 = fgetc(fn); \			
51	xval.lcc_b2 = fgetc(fn); \			
52	xval.lcc_b1 = fgetc(fn); \			
53	xval.lcc_b0 = fgetc(fn); \			
54	if (feof(fn) != 0) { \			
55	lm_queue_message(ERROR_MSG, "save-set file is corrupted"); \			
56	(void)fclose(fn); \			
57	return(LM_ERROR); \			
58	} \			
59	uval = xval.lcc_long; \			
60	#define read_short(fn, uval) { CONVERSION_RECORD xval; \			
61	xval.lcc_b1 = fgetc(fn); \			
62	xval.lcc_b0 = fgetc(fn); \			
63	if (feof(fn) != 0) { \			
64	lm_queue_message(ERROR_MSG, "save-set file is corrupted"); \			
65	(void)fclose(fn); \			
66	return(LM_ERROR); \			
67	} \			
68	uval = xval.lcc_short; \			
69	#define read_char(fn, uval) { uval = fgetc(fn); \			
70	if (feof(fn) != 0) { \			
71	lm_queue_message(ERROR_MSG, "save-set file is corrupted"); \			
72	(void)fclose(fn); \			
73	return(LM_ERROR); \			
74	} \			
75	#define print_long(file, uval) { read_long(file, uval); \			
76	printf("long 0x%08X\n", uval); }			
77	#define print_short(file, uval) { read_short(file, uval); \			
78	printf("short 0x%04X\n", uval); }			
79	#define print_char(file, uval) { read_char(file, uval); \			
80	printf("char 0x%02X\n", uval); }			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/futil.c	DATE 5/23/89	PAGE # 1/5
LINE #		SOURCE TEXT		
1		/* SCCS ID: futil.c rev 3.2, 5/9/89 at 16:26:26 */		
2		#include <stdio.h>		
3		#include <sys/types.h>		
4		#include <sys/stat.h>		
5		#include "common.h"		
6		#include "conv.h"		
7		#include "message.h"		
8		#include "network.h"		
9		#include "lm_sfi.h"		
10		#include "lzf.h"		
11		#include "lzf.h"		
12		#ifdef PC386		
13		#include <sys/file.h>		
14		/* PC386 */		
15		#include <fcntl.h>		
16		#include <io.h>		
17		/* PC386 */		
18		#endif		
19		#ifdef VMS		
20		#include <fcntl.h>		
21		#include <fcntl.h>		
22		#include <fcntl.h>		
23		#include <fcntl.h>		
24		#define BLOCK_SIZE 512		
25		/* VMS */		
26		#endif		
27		extern char *lm_malloc();		
28		extern char *lm_malloc();		
29		extern char *lm_malloc();		
30		lm_write_file_size(save_set, filename)		
31		FILE *save_set;		
32		char *filename;		
33		{		
34		int fd;		
35		struct stat file_info;		
36		long file_size;		
37		(void) fflush(save_set);		
38		fd = fileno(save_set);		
39		if (fstat(fd, &file_info) == -1) {		
40		lm_queue_message(ERROR_MSG, "fstat on filename: %s failed",		
41		filename);		
42		return (LM_ERROR);		
43		file_size = (long) file_info.st_size;		
44		/* The 0 test is required for BSD and SYS compatibility */		
45		if (fseek(save_set, (long) 0) != 0) {		
46		lm_queue_message(ERROR_MSG, "fseek on filename: %s failed",		
47		filename);		
48		return (LM_ERROR);		
49		write_long(save_set, file_size);		
50		return (LM_SUCCESS);		
51		}		
52		lm_check_file_size(save_set, filename)		
53		FILE *save_set;		
54		char *filename;		
55		{		
56		int fd;		
57		struct stat file_info;		
58		long file_size;		
59		fd = fileno(save_set);		
60		if (fstat(fd, &file_info) == -1) {		
61		lm_queue_message(ERROR_MSG, "fstat on filename: %s failed",		
62		filename);		
63		(void) fclose(save_set);		
64		return (LM_ERROR);		
65		read_long(save_set, &file_size);		
66		if (file_size != file_info.st_size) {		
67		lm_queue_message(ERROR_MSG, "save-set file: %s has been modified",		
68		filename);		
69		(void) fclose(save_set);		
70		return (LM_ERROR);		
71		return (LM_SUCCESS);		
72		}		
73		lm_get_file_size(file_ptr, filename, filesize)		
74		FILE *file_ptr;		
75		char *filename;		
76		long *filesize;		
77		{		
78		int fd;		
79		struct stat file_info;		
80		(void) fflush(file_ptr);		
81		fd = fileno(file_ptr);		
82		if (fstat(fd, &file_info) == -1) {		
83		lm_queue_message(ERROR_MSG, "fstat on filename: %s failed",		
84		filename);		
85		(void) fclose(file_ptr);		
86		return (FAILURE);		
87		*filesize = file_info.st_size;		
88		return (SUCCESS);		
89		}		
90		lm_check_file_type(filename)		
91		char *filename;		
92		{		
93		FILE *infile;		
94		struct stat file_info;		
95		int fd;		
96		u_short ftype;		
97		infile = fopen(filename, "r");		
98		/*		
99		* If file is not found then return SUCCESS. The fact that the file does not		
100		* exist will be checked later.		
101		*/		
102		if (infile == NULL)		
103		{		
104		return (SUCCESS);		
105		}		
106		return (LM_SUCCESS);		
107		}		
108		}		
109		}		
110		}		
111		}		
112		}		
113		}		
114		}		
115		}		
116		}		
117		}		
118		}		
119		}		
120		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/futil.c	DATE 5/23/89	PAGE # 2/6
LINE #		SOURCE TEXT		
221		return (SUCCESS);		
222		}		
223		id = fileinfo(infile);		
224		if (id != 0, &file_info == -1) {		
225		in_queue_message(ERROR_MSG, "stat on filename: %s failed",		
226		filename);		
227		return (FAILURE);		
228		}		
229		ftype = file_info.st_mode;		
230		(void) fclose(infile);		
231		}		
232		if (ftype & S_IFDIR) {		
233		in_queue_message(ERROR_MSG, "filename: %s is a directory",		
234		filename);		
235		return (FAILURE);		
236		}		
237		return (SUCCESS);		
238		}		
239		}		
240		in_search_key_id(head, instance_id, file_ptr)		
241		FILE_INFO *head;		
242		u_short instance_id;		
243		FILE_INFO **file_ptr;		
244		{		
245		while (head != NULL) {		
246		if (head->instance_id == instance_id)		
247		break;		
248		head = head->next;		
249		}		
250		if (head != NULL) {		
251		*file_ptr = head;		
252		return (SUCCESS);		
253		} else		
254		return (FAILURE);		
255		}		
256		}		
257		}		
258		in_search_key_name(head, filename)		
259		FILE_INFO *head;		
260		char *filename;		
261		{		
262		while (head != NULL) {		
263		if (strcmp(head->name, filename) == 0)		
264		break;		
265		head = head->next;		
266		}		
267		if (head != NULL)		
268		return (SUCCESS);		
269		else		
270		return (FAILURE);		
271		}		
272		}		
273		in_close_and_delete_file_key_id(head, instance_id)		
274		FILE_INFO *head;		
275		u_short instance_id;		
276		{		
277		FILE_INFO *ptr;		
278		FILE_INFO *prev_ptr;		
279		ptr = head;		
280		prev_ptr = NULL;		
281		while (ptr != NULL) {		
282		if (ptr->instance_id == instance_id) {		
283		if (prev_ptr == NULL) {		
284		head = ptr->next;		
285		} else {		
286		prev_ptr->next = ptr->next;		
287		}		
288		(void) fclose(ptr->file_ptr);		
289		free(ptr->name);		
290		free((char *) ptr);		
291		return;		
292		prev_ptr = ptr;		
293		ptr = ptr->next;		
294		}		
295		}		
296		}		
297		}		
298		in_open_and_insert_file(head, filename, instance_id)		
299		FILE_INFO *head;		
300		char *filename;		
301		u_short instance_id;		
302		{		
303		FILE *file;		
304		char *temp_name;		
305		FILE_INFO *file_info;		
306		temp_name = MALLOC((unsigned) (strlen(filename) + 1));		
307		if (temp_name == NULL) {		
308		in_queue_message(ERROR_MSG, "out of memory on host");		
309		return (FAILURE);		
310		}		
311		(void) strcpy(temp_name, filename);		
312		file_info = (FILE_INFO *) MALLOC(sizeof(FILE_INFO));		
313		if (file_info == NULL) {		
314		in_queue_message(ERROR_MSG, "out of memory on host");		
315		free(temp_name);		
316		return (FAILURE);		
317		}		
318		file = fopen(filename, "w");		
319		if (file == NULL) {		
320		in_queue_message(ERROR_MSG, "cannot open filename: %s for write",		
321		filename);		
322		free(temp_name);		
323		free((char *) file_info);		
324		return (FAILURE);		
325		}		
326		file_info->name = temp_name;		
327		file_info->file_ptr = file;		
328		file_info->instance_id = instance_id;		
329		file_info->next = head;		
330		head = file_info;		
331		}		
332		return (SUCCESS);		
333		}		
334		}		
335		/*=====		
336		*/		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM	DATE	PAGE #
		sfi/futil.c	TIME	3/7
LINE #	SOURCE TEXT			
241	* NAME:			
242	* lm_truncate -- Truncate a file to a certain size.			
243	* PURPOSE:			
244	* "lm_truncate" will truncate the requested file to the requested			
245	* size. If the file size is not greater than this, nothing is done.			
246	* INTERFACE:			
247	* lm_truncate(file_name, size);			
248	* PARAMETER TYPE DESCRIPTION			
249	* file_name char The name of the file to truncate.			
250	* size u_long The maximum size of the file after			
251	* truncating.			
252	* CALLS:			
253	* System calls "open", "close", "unlink" and "fstat".			
254	* C library call "sprintf".			
255	* EXTERNAL REFERENCES:			
256	* None.			
257	* RESTRICTIONS:			
258	* Should have write capability on source file.			
259	* RETURNS:			
260	* SUCCESS: If all goes well.			
261	* FAILURE: If something goes wrong.			
262	* DESIGNER:			
263	* Steve Parks			
264	*			
265	static			
266	lm_truncate(file_name, size)			
267	char *file_name,			
268	u_long size;			
269	#ifdef VMS			
270	{			
271	#ifdef APOLLO			
272	int From_FD; /* APOLLO */			
273	#endif			
274	#ifdef SUN			
275	int From_FD; /* SUN */			
276	#endif			
277	#ifdef PC186			
278	int From_FD; /* PC186 */			
279	#endif			
280	struct stat buf;			
281	/*			
282	* Get a file descriptor for the file to truncate.			
283	*/			
284	From_FD = open(file_name, O_RDWR, 0);			
285	if (From_FD < 0) {			
286	return (FAILURE);			
287	}			
288	/*			
289	* Get some information about the source file.			
290	*/			
291	if (fstat(From_FD, &buf) != 0) {			
292	(void) close(From_FD);			
293	return (FAILURE);			
294	}			
295	/*			
296	* If the file is already the right size, so need to continue.			
297	*/			
298	if ((u_long) buf.st_size > size) {			
299	#ifdef APOLLO			
300	if (ftruncate(From_FD, size) != 0) {			
301	(void) close(From_FD);			
302	return (FAILURE);			
303	}			
304	#endif			
305	/* APOLLO */			
306	#ifdef SUN			
307	if (ftruncate(From_FD, size) != 0) {			
308	(void) close(From_FD);			
309	return (FAILURE);			
310	}			
311	#endif			
312	/* SUN */			
313	#ifdef PC186			
314	if (chsize(From_FD, size) != 0) {			
315	(void) close(From_FD);			
316	return (FAILURE);			
317	}			
318	#endif			
319	/* PC186 */			
320	{			
321	(void) close(From_FD);			
322	return (SUCCESS);			
323	}			
324	#else			
325	/* VMS */			
326	{			
327	struct RAB record_access_block;			
328	struct FAB file_access_block;			
329	char file_buffer[BLOCK_SIZE];			
330	int xms_status;			
331	int total_bytes;			
332	record_access_block = ccirms_rab;			
333	file_access_block = ccirms_fab;			
334	record_access_block.rab\$1_fab = &file_access_block;			
335	file_access_block.fab\$1_fab = file_name;			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/futil.c	DATE 5/23/89	PAGE # 4/8
SOURCE TEXT				
LINE #				
351	file_access_block.fabbb_fas = strlen(file_access_block.fabbl_fas);			
352	file_access_block.fabbb_fac = FABSM_CST;			
353	file_access_block.fabbb_fac = FABSM_TWS;			
354	file_access_block.fabbb_mrs = BLOCK_SIZE;			
355	file_access_block.fabbb_ory = FABSM_SEQ;			
356	rma_status = SYSOPEN(&file_access_block);			
357	if ((rma_status != SYS_NORMAL) && (rma_status != RMNS_NORMAL)) {			
358	ln_queue_message(VMS_ERROR_MSG, rma_status, "Unable to open file %s", file_name);			
359	return (FAILURE);			
360	}			
361	rma_status = SYSCONNECT(&record_access_block);			
362	if ((rma_status != SYS_NORMAL) && (rma_status != RMNS_NORMAL)) {			
363	ln_queue_message(VMS_ERROR_MSG, rma_status, "Unable to connect to file %s", file_name);			
364	return (FAILURE);			
365	}			
366	total_bytes = 0;			
367	while (total_bytes <= size) {			
368	record_access_block.rabbb_rac = FABSM_SEQ;			
369	record_access_block.rabbb_rbf = file_buffer;			
370	record_access_block.rabbb_rsz = sizeof(file_buffer);			
371	rma_status = SYSFIND(&record_access_block);			
372	if ((rma_status != SYS_NORMAL) && (rma_status != RMNS_NORMAL)) {			
373	ln_queue_message(VMS_ERROR_MSG, rma_status, "Unable to read file %s", file_name);			
374	return (FAILURE);			
375	}			
376	total_bytes += strlen(file_buffer) + 1;			
377	}			
378	rma_status = SYSSTRUNCATE(&record_access_block);			
379	if ((rma_status != SYS_NORMAL) && (rma_status != RMNS_NORMAL)) {			
380	ln_queue_message(VMS_ERROR_MSG, rma_status, "Unable to truncate file %s", file_name);			
381	return (FAILURE);			
382	}			
383	rma_status = SYSCLOSE(&file_access_block);			
384	if ((rma_status != SYS_NORMAL) && (rma_status != RMNS_NORMAL)) {			
385	ln_queue_message(VMS_ERROR_MSG, rma_status, "Unable to close file %s", file_name);			
386	return (FAILURE);			
387	}			
388	return (SUCCESS);			
389	}			
390	#endif			
391	ln_append_and_insert_file(head, filename, instance_id, filesize)			
392	FILE_INFO			
393	char			
394	u_short			
395	u_long			
396	{			
397	filename;			
398	instance_id;			
399	filesize;			
400	}			
401	/*			
402	This routine opens "filename" for append and link the file information to			
403	the FILE_INFO link list. It truncates the file to at most filesize bytes.			
404	*/			
405	FILE			
406	char			
407	FILE_INFO			
408	long			
409	int			
410	{			
411	temp_name = DALLOC((unsigned) (strlen(filename) + 1));			
412	if (temp_name == NULL) {			
413	ln_queue_message(ERROR_MSG, "out of memory on host");			
414	return (FAILURE);			
415	}			
416	(void) strcpy(temp_name, filename);			
417	file_info = (FILE_INFO *) DALLOC((unsigned) sizeof(FILE_INFO));			
418	if (file_info == NULL) {			
419	ln_queue_message(ERROR_MSG, "out of memory on host");			
420	free(temp_name);			
421	return (FAILURE);			
422	}			
423	accessible = (long) access(filename, (int) 0);			
424	if (accessible != 0) {			
425	ln_queue_message(WARNING_MSG, "filename: %s not found, can't continue logging",			
426	filename);			
427	free(temp_name);			
428	free((char *) file_info);			
429	return (SUCCESS);			
430	}			
431	if (ln_truncate(filename, filesize) != SUCCESS) {			
432	ln_queue_message(WARNING_MSG, "cannot truncate filename: \"%s\", cannot continue logging",			
433	filename);			
434	free(temp_name);			
435	free((char *) file_info);			
436	return (SUCCESS);			
437	}			
438	file = fopen(filename, "a");			
439	if (file == NULL) {			
440	ln_queue_message(WARNING_MSG, "cannot open filename: \"%s\" for append, cannot continue logging",			
441	filename);			
442	free(temp_name);			
443	free((char *) file_info);			
444	return (SUCCESS);			
445	}			
446	file_info->name = temp_name;			
447	file_info->file_ptr = file;			
448	file_info->instance_id = instance_id;			
449	file_info->next = "head";			
450	"head" = file_info;			
451	return (SUCCESS);			
452	}			
453	#endif			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/globals.c	DATE 5/23/89 TIME 1:20:51 pm	PAGE # 1/9
LINE #	SOURCE TEXT			
1	/* SOCS ID: globals.c rev 3.1, 4/24/89 at 07:35:46 */			
2	#include <stdio.h>			
3	#include <ctype.h>			
4	#include "common.h"			
5	#include "network.h"			
6	#include "messages.h"			
7	#include "lm_sfi.h"			
8	#include "lafi.h"			
9	#include "protans.h"			
10	#include "protcmd.h"			
11	#include "conv.h"			
12	/* globals from sfi.c */			
13	/* simulation tables */			
14	SYSTEM_INFO *lm_system_table, /* ptr to system array */			
15	DEFINITION_INFO *lm_def_id_table, /* ptr to def id array */			
16	INSTANCE_INFO *lm_inst_id_table, /* ptr to instance id array */			
17	INSTANCE_INFO *lm_free_instance_id,			
18	u_short lm_def_id_table_size,			
19	u_short lm_inst_id_table_size,			
20	u_short lm_def_id_avail,			
21	/* Pointers to Entities selected */			
22	SYSTEM_INFO *lm_system_selected,			
23	DEFINITION_INFO *lm_definition_selected,			
24	INSTANCE_INFO *lm_instance_selected,			
25	u_long lm_last_definition_id_specified,			
26	u_long lm_last_instance_id_specified,			
27	/* Logic level mapping */			
28	USER_TO_LM_LOGIC_LEVEL lm_user_to_lm_logic_level_map[MAX_LOGIC_STATE],			
29	u_short lm_to_user_logic_level_map[MAX_OUTPUT_LOGIC_SIZE],			
30	u_short lm_logic_level_map_count,			
31	/* File processing */			
32	FILE_INFO *lm_vector_file_ptr,			
33	FILE_INFO *lm_timing_file_ptr,			
34	/* Temporary buffer for DASH processing */			
35	char *lm_tbuf_buffer,			
36	char *lm_tbuf_ptr,			
37	char *lm_max_tbuf_addr,			
38	/* simulation characteristics */			
39	u_short lm_sim_verified,			
40	u_long lm_gbl_simulator_type,			
41	u_long lm_gbl_time_scale_number,			
42	u_long lm_gbl_time_scale_units,			
43	u_long lm_gbl_divide_delay,			
44	u_long lm_gbl_time_scale,			
45	u_long lm_gbl_half_time_scale,			
46	u_long lm_simulation_time,			
47	u_short lm_host_fileopened,			
48	#ifdef DEBUG			
49	u_char lm_debug[MAX_DEBUG],			
50	#endif			
51	char lm_gbl_username[MAX_STRING_LENGTH],			
52	char lm_gbl_hostname[MAX_STRING_LENGTH],			
53	/* sfi.c former static variables */			
54	u_short lm_get_ptr_value_type,			
55	u_short lm_first_call,			
56	FILE *lm_save_set,			
57	char *lm_modeler_list_buf[MAX_SYSTEM_TABLE_SIZE],			
58	long lm_iul_users[MAX_USER_COUNT],			
59	char *lm_iul_hostnames[MAX_USER_COUNT],			
60	char *lm_iul_usernames[MAX_USER_COUNT],			
61	char lm_user_addr[MAX_USER_COUNT * MAX_STRING_LENGTH],			
62	char lm_user_addr[MAX_USER_COUNT * MAX_STRING_LENGTH],			
63	long lm_idl_device_numbers[MAX_LANE_COUNT * MAX_SLOT_COUNT],			
64	char *lm_idl_device_names[MAX_LANE_COUNT * MAX_SLOT_COUNT],			
65	char lm_idl_buf[MAX_LANE_COUNT * MAX_SLOT_COUNT * MAX_STRING_LENGTH],			
66	char lm_gn_error_string[MAX_MESSAGE],			
67	char lm_cdf_modeler_name[MAX_STRING_LENGTH],			
68	char lm_cdf_device_name[MAX_STRING_LENGTH],			
69	char lm_crd_modeler_name[MAX_STRING_LENGTH],			
70	char lm_crd_device_name[MAX_STRING_LENGTH],			
71	char lm_id_string[MAX_STRING_LENGTH],			
72	long lm_ida_lanes[MAX_LANE_COUNT * MAX_SLOT_COUNT],			
73	long lm_ida_slots[MAX_LANE_COUNT * MAX_SLOT_COUNT],			
74	char lm_idp_string[MAX_STRING_LENGTH],			
75	char lm_idp_stringl[MAX_STRING_LENGTH],			
76	char lm_idp_stringl[MAX_STRING_LENGTH],			
77	/* access.c former static variables */			
78	char lm_modeler_name_selected[MAX_STRING_LENGTH],			
79	short lm_modeler_id,			
80	/* network.c global variables */			
81	CONNECTION *lm_table_of_conns[MAX_SYSTEM_TABLE_SIZE + 1],			
82	CONNECTION *lm_global_conn_ptr,			
83	/* timer.c global variables */			
84	u_long lm_tick,			
85	lm_init_globals(simulator_type)			
86	u_long simulator_type,			
87	{			
88	lm_gbl_simulator_type = simulator_type,			
89	/* initialize global variables */			
90	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/globals.c	DATE 5/23/89	PAGE # 2/10
LINE #		SOURCE TEXT		
21		lm_last_definition_id_specified = -1,		
22		lm_last_instance_id_specified = -1,		
23		lm_logic_level_max_count = MAX_LOGIC_STATE + 1,		
24		lm_vector_file_ptr = NULL,		
25		lm_timing_file_ptr = NULL,		
26		lm_sim_verified = 0,		
27		lm_gbl_time_scale_number = -1,		
28		lm_host_file_opened = 0,		
29		lm_get_pin_value_type = -1,		
30				
31		lm_simulation_time = (u_long) 0,		
32				
33		(void)strcpy(lm_gbl_username, "mouser");		
34		(void)strcpy(lm_gbl_hostname, "mohost");		
35				
36		lm_first_call = TRUE,		
37		lm_save_set = NULL,		
38				
39		lm_init_lists(),		
40				
41		/* from access.c */		
42				
43		(void) strcpy(lm_modeler_name_selected, "(none)"),		
44		lm_modeler_id = -1,		
45				
46		/* from timer.c */		
47				
48		lm_tick = 0,		
49		}		
50				
51		lm_init_lists()		
52		{		
53		register int i,		
54				
55		/* sfi.c: */		
56		/* lm_inquire_modeler_list: */		
57				
58		for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i)		
59		lm_modeler_list_buf[i] = NULL,		
60				
61		/* lm_inquire_user_list: */		
62				
63		for (i = 0; i < MAX_USER_COUNT; ++i) {		
64		lm_iul_hostnames[i] = &(lm_user_hbuf[i * MAX_STRING_LENGTH]),		
65		lm_iul_usernames[i] = &(lm_user_ubuf[i * MAX_STRING_LENGTH]),		
66		}		
67				
68		/* lm_inquire_device_list: */		
69				
70		for (i = 0; i < MAX_LANE_COUNT * MAX_SLOT_COUNT; ++i) {		
71		lm_idl_device_names[i] = &(lm_idl_buf[i * MAX_STRING_LENGTH]),		
72		}		
73				
74		/* network.c: */		
75				
76		lm_global_conn_ptr = 0,		
77		for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i)		
78		lm_table_of_conn[i] = NULL,		
79		}		

Copyright 1989 Logic Modeling Systems		HEADER FILE sfi/globals.h	DATE 5/23/89	PAGE # 1/11
LINE #	HEADER TEXT			
1	/* SCCI_ID: globals.h rev 3.1.1 4/24/89 at 07:35:49 */			
2	/* globals from sfi.c */			
3	/* simulation tables */			
4	extern SYSTEM_INFO *lm_system_table, /* ptr to system array */			
5	extern DEFINITION_INFO *lm_def_id_table, /* ptr to def id array */			
6	extern INSTANCE_INFO *lm_inst_id_table, /* ptr to instance id array */			
7	extern u_short lm_def_id_table_size,			
8	extern u_short lm_inst_id_table_size,			
9	extern u_short lm_def_id_avail,			
10	/* Pointers to Entities selected */			
11	extern SYSTEM_INFO *lm_system_selected,			
12	extern DEFINITION_INFO *lm_definition_selected,			
13	extern INSTANCE_INFO *lm_instance_selected,			
14	extern u_long lm_last_definition_id_specified,			
15	extern u_long lm_last_instance_id_specified,			
16	/* Logic level mapping */			
17	extern USER_TO_LM_LOGIC_LEVEL lm_user_to_lm_logic_level_map[],			
18	extern u_short lm_to_user_logic_level_map[],			
19	extern u_short lm_logic_level_map_count,			
20	/* File processing */			
21	extern FILE_INFO *lm_vector_file_ptr,			
22	extern FILE_INFO *lm_timing_file_ptr,			
23	/* Temporary buffer for NAME processing */			
24	extern char *lm_temp_buffer,			
25	extern char *lm_lm_ptr,			
26	extern char *lm_max_char_addr,			
27	/* Simulation characteristics */			
28	extern u_short lm_sim_verified,			
29	extern u_long lm_gbl_simulator_type,			
30	extern u_long lm_gbl_time_scale_number,			
31	extern u_long lm_gbl_time_scale_units,			
32	extern u_char lm_gbl_divide_delay,			
33	extern u_long lm_gbl_time_scale,			
34	extern u_long lm_gbl_half_time_scale,			
35	extern u_long lm_simulation_time,			
36	extern u_short lm_host_file_opened,			
37	#ifdef DEBUG			
38	extern u_char lm_debug[],			
39	#endif			
40	extern long lm_inpt_pin_count,			
41	extern long lm_outpt_pin_count,			
42	#ifdef DEBUG			
43	extern long lm_start_time,			
44	extern long lm_total_time,			
45	#endif			
46	extern char lm_gbl_username[],			
47	extern char lm_gbl_hostname[],			
48	/* sfi.c former static variables */			
49	extern short lm_get_pin_value_type,			
50	extern u_short lm_first_call,			
51	extern FILE *lm_save_out,			
52	extern char *lm_modeler_list_buf[],			
53	extern long lm_idl_users[],			
54	extern char *lm_idl_hostnames[],			
55	extern char *lm_idl_usernames[],			
56	extern char lm_user_hbuf[],			
57	extern char lm_user_ubuf[],			
58	extern long lm_idl_device_numbers[],			
59	extern char *lm_idl_device_names[],			
60	extern char lm_idl_buf[],			
61	extern char lm_gn_error_string[],			
62	extern char lm_cdf_modeler_name[],			
63	extern char lm_cdf_device_name[],			
64	extern char lm_crd_modeler_name[],			
65	extern char lm_crd_device_name[],			
66	extern char lm_ind_string[],			
67	extern long lm_ida_names[],			
68	extern long lm_ida_slots[],			
69	extern char lm_idp_string[],			
70	extern char lm_idp_string2[],			
71	extern char lm_idp_string3[],			
72	/* access.c former static variables */			
73	extern char lm_modeler_name_selected[],			
74	extern short lm_modeler_id,			
75	/* network.c global variables */			
76	extern CONNECTION *lm_table_of_coasms[],			
77	extern CONNECTION *lm_global_coas_ptr,			
78	/* timer.c global variables */			
79	extern u_long lm_tick,			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89 TIME 1:20:52 pm	PAGE # 1/12
LINE #	SOURCE TEXT			
1	/* SCSF_ID: lmtest.c rev 3.2, 5/9/89 at 15:32:50 */			
2	/*			
3	LM-1000 Test Vector Player			
4	*/			
5	usage:			
6	-i play test vectors			
7	-i instance or fault id			
8	-t test vector file			
9	-d discrepancy report file			
10	-o discrepancy cutoff			
11	-n number of nets			
12	-g number of gates			
13	-s number of time settings			
14	-p number of pin map settings			
15	-m number of discrepancies			
16	-l long instance or fault id			
17	-c char			
18	-t test vector file			
19	-d discrepancy report file			
20	-o unsigned long discrepancy cutoff			
21	-n unsigned long			
22	-g number of nets			
23	-s number of gates			
24	-p number of time settings			
25	-m number of pin map settings			
26	-l number of discrepancies			
27	-c instance or fault id: id of instance or fault to which to play vectors			
28	-t test vector file: name of file containing test vectors to play (".TST")			
29	-i if LM_NO_FILE is passed, test vectors are taken from stdin			
30	-d discrepancy report file: name of file to which to write discrepancies			
31	-o if LM_NO_FILE is passed, discrepancies are written to stdout			
32	-o discrepancy cutoff: count of number of discrepancies before quitting			
33	-n use 0 to indicate never quit			
34	-g number of nets: number of times lm_set_pin_value() was called			
35	-s number of gates: number of times lm_get_pin_change() was called			
36	-p number of time settings: number of calls to lm_set_simulation_time()			
37	-m number of pin map settings: number of calls to lm_set_pin_map()			
38	-l number of discrepancies: number of discrepancy vectors			
39	-c resolve test vector filenames unless LM_NO_FILE is passed:			
40	-t lm_resolve_library_file("LM_LIB", test_vector_file, tv_is, &discrep(tv_is))			
41	-i requires the associated definition id of the passed instance:			
42	-d lm_inquire_instance(instance, LM_ASSOCIATED_DEFINITION, &definition)			
43	-o note the pin map of the definition using test vector file header:			
44	-p lm_set_pin_map(&definition, pin_name, LM_DONT_CARE, column-1)			
45	-m for each pin_name/column pair in the header			
46	-l plays vectors using the instance and the rest of test vector stream:			
47	-t lm_set_simulation_time((unsigned long)0, timestamp)			
48	-p lm_get_pin_change()			
49	-m lm_set_pin_value()			
50	-c returns LM_PROCESSED if no warnings or errors			
51	-t otherwise returns LM_WARNING if no errors			
52	-o otherwise returns LM_ERROR			
53	*/			
54	#include <stdio.h>			
55	#include "common.h"			
56	#include "message.h"			
57	#include "lm_sfi.h"			
58	#define USE_MACROS			
59	#define MAX_PIN_COUNT 640			
60	#define MAX_TOKEN_LENGTH 1024			
61	#define MAX_TV_FILENAME_LENGTH 256			
62	#define LM_LIBRARY_VARIABLE_NAME "LM_LIB"			
63	char lm_obscure_dont_set_pinmap_flag = 0;			
64	static char tok [MAX_TOKEN_LENGTH];			
65	static char pin [MAX_TOKEN_LENGTH];			
66	static char tv_filename [MAX_TV_FILENAME_LENGTH];			
67	static unsigned long			
68	timestamp,			
69	set_count,			
70	get_count,			
71	vector_count,			
72	pin_count,			
73	disc_count,			
74	static FILE			
75	"tv_file",			
76	"dr_file",			
77	static long definition_id,			
78	static struct test_pin			
79	{ long in_val, out_val,			
80	char tv_pin, is_in, is_out,			
81	} test_pins [MAX_PIN_COUNT];			
82	static unsigned short			
83	lexerr (str)			
84	register char *str,			
85	{ lm_queue_message			
86	(ERROR_MSG,			
87	"test vector lexical error: %s",			
88	str			
89	},			
90	return 0;			
91	} /* lexerr */			
92	static unsigned short			
93	illchar (c)			
94	register char c,			
95	{ lm_queue_message			
96	(ERROR_MSG,			
97	"test vector lexical error: illegal character: '%c'",			
98	c			
99	},			
100	return 0;			
101	} /* illchar */			
102	static unsigned short			
103	ccccc ()			
104	/* scan a (character) token from tv_file, store it in tok */			
105	{ register short nextc;			
106	nextc =getc (tv_file);			
107	while (nextc != EOF)			
108	switch (nextc)			
109	{			
110	}			
111	}			
112	}			
113	}			
114	}			
115	}			
116	}			
117	}			
118	}			
119	}			
120	}			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/lmtest.c

DATE 5/23/89
TIME 1:20:52 pm

PAGE #
2/13

```

LINE # SOURCE TEXT
121 { case '!' :
122 while ( ( nextc = getc ( tv_file ) ) != EOF && nextc != '\n' ) ,
123 if ( nextc == EOF )
124 return lerror ( "unterminated comment" ) ,
125 /* Fall through */
126 case 'D' :
127 case 'U' :
128 case 'M' :
129 case 'I' :
130 case 'T' :
131 case 'X' :
132 case 'T' :
133 case 'O' :
134 case 'I' :
135 case '7' :
136 case 'F' :
137 case 'J' :
138 case '\n' :
139 *tok = nextc ;
140 tok [ 1 ] = '\0' ;
141 return 1 ;
142 default :
143 if ( nextc >= '\0' && nextc <= ' ' )
144 { while
145 ( ( nextc = getc ( tv_file ) ) >= '\0' &&
146 nextc <= ' ' && nextc != '\n' )
147 } ,
148 }
149 else
150 return iluchar ( nextc ) ,
151 break ;
152 }
153 *tok = '\0' ;
154 return 1 ;
155 /* hscan */
156
157 static unsigned short
158 hscan ( )
159 /* scan a (header) token from tv_file, store it in tok */
160 { register short nextc ;
161 register char *t ;
162 nextc = getc ( tv_file ) ;
163 while ( nextc != EOF )
164 switch ( nextc )
165 { case '!' :
166 while
167 ( ( nextc = getc ( tv_file ) ) != EOF &&
168 nextc != '\n' && nextc >= '\0' && nextc <= ' ' )
169 } ,
170 if ( nextc != EOF && nextc != '\n' )
171 { if ( nextc == 't' )
172 { t = tok ;
173 *t++ = nextc ;
174 nextc = getc ( tv_file ) ;
175 if ( nextc == 'e' )
176 { *t++ = nextc ;
177 nextc = getc ( tv_file ) ;
178 if ( nextc == 's' )
179 { *t++ = nextc ;
180 nextc = getc ( tv_file ) ;
181 if ( nextc == 't' )
182 { *t++ = nextc ;
183 while
184 ( ( nextc = getc ( tv_file ) ) != EOF &&
185 nextc != '\n' )
186 } ,
187 if ( nextc == EOF )
188 return lerror ( "unterminated comment" ) ;
189 *t = '\0' ;
190 return 1 ;
191 }
192 }
193 }
194 }
195 }
196 if ( nextc != EOF && nextc != '\n' )
197 while
198 ( ( nextc = getc ( tv_file ) ) != EOF && nextc != '\n' )
199 } ,
200 }
201 if ( nextc == EOF )
202 return lerror ( "unterminated comment" ) ,
203 /* Fall through */
204 case 'I' :
205 case 'O' :
206 case 'I' :
207 case '\n' :
208 *tok = nextc ;
209 tok [ 1 ] = '\0' ;
210 return 1 ;
211 default :
212 if ( nextc >= '0' && nextc <= '9' )
213 { t = tok ;
214 *t++ = nextc ;
215 while ( ( nextc = getc ( tv_file ) ) >= '0' && nextc <= '9' )
216 *t++ = nextc ;
217 ( void ) ungetc ( nextc , tv_file ) ;
218 *t = '\0' ;
219 return 1 ;
220 }
221 else if ( nextc >= '\0' && nextc <= ' ' )
222 { while
223 ( ( nextc = getc ( tv_file ) ) >= '\0' &&
224 nextc <= ' ' && nextc != '\n' )
225 } ,
226 }
227 else
228 return iluchar ( nextc ) ,
229 break ;
230 }
231 *tok = '\0' ;
232 return 1 ;
233 /* hscan */
234
235 static unsigned short
236 hscan ( )
237 /* scan a (signalname) token from tv_file, store it in tok */
238 { register short nextc ;
239 register char *t ;
240 nextc = getc ( tv_file ) ;

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 3/14
LINE #	SOURCE TEXT			
241	while (nextc != EOF)			
242	switch (nextc)			
243	{ case ' ':			
244	while			
245	{ (nextc = getc (tv_file)) != EOF &&			
246	nextc != '\n' && nextc != '\0' && nextc != ' ' }			
247	}			
248	if (nextc != EOF && nextc != '\n')			
249	{ if (nextc == 'p')			
250	{ t = tok ;			
251	{ t++ = nextc ;			
252	nextc = getc (tv_file) ;			
253	if (nextc == 'a')			
254	{ t++ = nextc ;			
255	nextc = getc (tv_file) ;			
256	if (nextc == 't')			
257	{ t++ = nextc ;			
258	nextc = getc (tv_file) ;			
259	if (nextc == 'c')			
260	{ t++ = nextc ;			
261	nextc = getc (tv_file) ;			
262	if (nextc == 'e')			
263	{ t++ = nextc ;			
264	nextc = getc (tv_file) ;			
265	if (nextc == 'x')			
266	{ t++ = nextc ;			
267	nextc = getc (tv_file) ;			
268	if (nextc == 's')			
269	{ t++ = nextc ;			
270	nextc = getc (tv_file) ;			
271	if (nextc == 's')			
272	{ t++ = nextc ;			
273	while			
274	{ (nextc = getc (tv_file)			
275) != EOF &&			
276	nextc != '\n' }			
277	if (nextc == EOF)			
278	return			
279	lerror			
280	{ "unterminated comment"			
281	}			
282	t = '\0' ;			
283	return 1 ;			
284	}			
285	}			
286	}			
287	}			
288	}			
289	}			
290	}			
291	}			
292	}			
293	}			
294	}			
295	}			
296	}			
297	}			
298	}			
299	}			
300	}			
301	}			
302	}			
303	}			
304	}			
305	}			
306	}			
307	}			
308	}			
309	}			
310	}			
311	}			
312	}			
313	}			
314	}			
315	}			
316	}			
317	}			
318	}			
319	}			
320	}			
321	}			
322	}			
323	}			
324	}			
325	}			
326	}			
327	}			
328	}			
329	}			
330	}			
331	}			
332	}			
333	}			
334	}			
335	}			
336	}			
337	}			
338	}			
339	}			
340	}			
341	}			
342	}			
343	}			
344	}			
345	}			
346	}			
347	}			
348	}			
349	}			
350	}			
351	}			
352	}			
353	}			
354	}			
355	}			
356	}			
357	}			
358	}			
359	}			
360	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 4/15
LINE #		SOURCE TEXT		
361		nextc =getc (tv_file) ;		
362		if (nextc != '\n')		
363		return lerror ("newline expected after backslash") ;		
364		nextc =getc (tv_file) ;		
365		break ;		
366		default :		
367		if (nextc >= '0' && nextc <= '9')		
368		{		
369		t = tok ;		
370		while ((nextc =getc (tv_file)) >= '0' && nextc <= '9')		
371		t++ = nextc ;		
372		if (nextc == '.')		
373		{		
374		t++ = nextc ;		
375		while		
376		{ (nextc =getc (tv_file)) >= '0' && nextc <= '9'		
377		{		
378		t++ = nextc ;		
379		(void) ungetc (nextc , tv_file) ;		
380		}		
381		else (void) ungetc (nextc , tv_file) ;		
382		t = '\0' ;		
383		return 1 ;		
384		}		
385		else if (nextc >= '\0' && nextc <= ' ')		
386		{ while		
387		{ (nextc =getc (tv_file)) >= '\0' &&		
388		nextc <= ' ' && nextc != '\n'		
389		{		
390		}		
391		else		
392		return illichar (nextc) ;		
393		break ;		
394		}		
395		tok = '\0' ;		
396		return 1 ;		
397		} /* vacan */		
398				
399		static unsigned short		
400		vacan ()		
401		/* scan a (vector) taken from tv_file, store it in tok */		
402		{ register short nextc ;		
403		nextc =getc (tv_file) ;		
404		while (nextc != EOF)		
405		switch (nextc)		
406		{ case '!' :		
407		while ((nextc =getc (tv_file)) != EOF && nextc != '\n') ,		
408		if (nextc == EOF)		
409		return lerror ("unterminated comment") ;		
410		/* Fall through */		
411		case 'D' :		
412		case 'U' :		
413		case 'N' :		
414		case 'Z' :		
415		case 'L' :		
416		case 'W' :		
417		case 'I' :		
418		case 'T' :		
419		case 'O' :		
420		case '1' :		
421		case '7' :		
422		case 'F' :		
423		case 'A' :		
424		tok = nextc ;		
425		tok [1] = '\0' ;		
426		return 1 ;		
427		case '\\ :		
428		nextc =getc (tv_file) ;		
429		if (nextc != '\n')		
430		return lerror ("newline expected after backslash") ;		
431		nextc =getc (tv_file) ;		
432		break ;		
433		default :		
434		if (nextc >= '\0' && nextc <= ' ')		
435		{ while		
436		{ (nextc =getc (tv_file)) >= '\0' &&		
437		nextc <= ' ' && nextc != '\n'		
438		{		
439		}		
440		else		
441		return illichar (nextc) ;		
442		break ;		
443		}		
444		tok = '\0' ;		
445		return 1 ;		
446		} /* vacan */		
447				
448		static long		
449		tv_error (str)		
450		register char *str ;		
451		{ lm_queue_message		
452		{ ERROR_MSG ,		
453		"test vector syntax error: %s" ,		
454		str		
455		} ,		
456		return LM_ERROR ,		
457		} /* tv_error */		
458				
459		static long		
460		tv_check (str)		
461		register char *str ;		
462		{ lm_queue_message		
463		{ ERROR_MSG ,		
464		"test vector constraint error: %s" ,		
465		str		
466		} ,		
467		return LM_ERROR ,		
468		} /* tv_check */		
469				
470		static long		
471		tv_internal (str)		
472		register char *str ;		
473		{ lm_queue_message		
474		{ ERROR_MSG ,		
475		"test vector internal error: %s" ,		
476		str		
477		} ,		
478		return LM_ERROR ,		
479		} /* tv_internal */		
480				

```

481 static unsigned long
482 value_of ( str )
483 {
484     register char *str ,
485     /* Returns the value of the passed string */
486     { register unsigned long ret = 0 ,
487     while ( *str != '\0' )
488     { if ( *str == '0' && *str <= '9' )
489     { ret = 10 ;
490     ret += *str - '0' ;
491     }
492     else
493     { return -1 ,
494     ++str ,
495     }
496     return ret ,
497     } /* value_of */
498
499 #ifdef USE_MACROS
500 #define is_name(str)((str)!='\0'&&(str)(<='\0'&&(str)!='\0'))
501 #else /* USE_MACROS */
502 static unsigned short
503 is_name ( str )
504 { char *str ,
505 { return *str == '\0' && *str <= '\0' && *str != '\0' ,
506 } /* is_name */
507 #endif /* USE_MACROS */
508
509 #ifdef USE_MACROS
510 #define is_column(str)((str)!='0'&&(str)(<='9'))
511 #else /* USE_MACROS */
512 static unsigned short
513 is_column ( str )
514 { char *str ,
515 { return *str == '0' && *str <= '9' ,
516 } /* is_column */
517 #endif /* USE_MACROS */
518
519 #ifdef USE_MACROS
520 #define is_state_char(str)(state_char_table[(str)&0xFF])
521 static char state_char_table [ 256 ] =
522 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 0 to 15 */
523 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 16 to 31 */
524 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 32 to 47 */
525 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, /* 48 to 63 */
526 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, /* 64 to 79 */
527 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 80 to 95 */
528 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 96 to 111 */
529 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 112 to 127 */
530 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 128 to 143 */
531 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 144 to 159 */
532 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 160 to 175 */
533 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 176 to 191 */
534 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 192 to 207 */
535 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 208 to 223 */
536 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 224 to 239 */
537 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 240 to 255 */
538 } ,
539 #endif /* USE_MACROS */
540 static unsigned short
541 is_state_char ( str )
542 { char *str ,
543 { switch ( *str )
544 { case 'D' :
545 case 'U' :
546 case 'W' :
547 case 'Z' :
548 case 'L' :
549 case 'H' :
550 case 'X' :
551 case 'T' :
552 case '0' :
553 case '1' :
554 case '2' :
555 case 'P' :
556 case 'F' :
557     return 1 ,
558     default : return 0 ,
559 } } /* is state_char */
560 #endif /* USE_MACROS */
561
562 #ifdef USE_MACROS
563 #define is_number(str)((str)!='0'&&(str)(<='9') || (str)!='.')
564 #else /* USE_MACROS */
565 static unsigned short
566 is_number ( str )
567 { char *str
568 { return *str == '0' && *str <= '9' || *str == '.' ,
569 } /* is number */
570 #endif /* USE_MACROS */
571
572 static double
573 float_value ( str )
574 { register char *str ,
575 { double val = 0 , power = 0 ,
576 char *s = str ,
577 while ( *str != '\0' )
578 { switch ( *str )
579 { case '.' :
580 if ( power )
581 { in_queue_message ( ERROR_MSG , "bad timestamp: %s" , s ) ,
582 return -1 ,
583 }
584 power = 10 ,
585 break ,
586 case '0' :
587 case '1' :
588 case '2' :
589 case '3' :
590 case '4' :
591 case '5' :
592 case '6' :
593 case '7' :
594 case '8' :
595 case '9' :
596 if ( power )
597 { val += ( *str - '0' ) / power ,
598 power *= 10 ,
599 }
600 else
601 { val *= 10 ,

```


Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 6/17
LINE #		SOURCE TEXT		
601		val += *str - '0' ;		
602		}		
603		break ;		
604		default :		
605		in_queue_message (ERROR_MSG , "bad timestamp: ts" , s) ,		
606		return -1 ;		
607		}		
608		++str ;		
609		}		
610		return val ;		
611		} /* float_value */		
612		}		
613		# ifdef USE_MACROS		
614		# define is_known_direction(c)(known_direction_table[(c)&0xFF])		
615		static char known_direction_table [256] =		
616		{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 0 to 15 */		
617		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 16 to 31 */		
618		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 32 to 47 */		
619		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 48 to 63 */		
620		0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, /* 64 to 79 */		
621		0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, /* 80 to 95 */		
622		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 96 to 111 */		
623		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 112 to 127 */		
624		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 128 to 143 */		
625		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 144 to 159 */		
626		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 160 to 175 */		
627		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 176 to 191 */		
628		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 192 to 207 */		
629		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 208 to 223 */		
630		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 224 to 239 */		
631		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 240 to 255 */		
632		}		
633		# else /* USE_MACROS */		
634		static unsigned short		
635		is_known_direction (c)		
636		char c ;		
637		{ switch (c)		
638		{ case 'D' :		
639		case 'U' :		
640		case 'Z' :		
641		case 'W' :		
642		case 'L' :		
643		case 'E' :		
644		case 'T' :		
645		case 'X' :		
646		return 1 ;		
647		default : return 0 ;		
648		}		
649		} /* is_known_direction */		
650		# endif /* USE_MACROS */		
651		# ifdef USE_MACROS		
652		# define is_sense(c)((c)=='L' (c)=='H' (c)=='T' (c)=='X')		
653		# else /* USE_MACROS */		
654		static unsigned short		
655		is_sense (c)		
656		char c ;		
657		{ switch (c)		
658		{ case 'L' :		
659		case 'H' :		
660		case 'T' :		
661		case 'X' :		
662		return 1 ;		
663		default : return 0 ;		
664		}		
665		} /* is_sense */		
666		# endif /* USE_MACROS */		
667		# ifdef USE_MACROS		
668		# define is_drive(c)((c)=='D' (c)=='U' (c)=='Z' (c)=='W')		
669		# else /* USE_MACROS */		
670		static unsigned short		
671		is_drive (c)		
672		char c ;		
673		{ switch (c)		
674		{ case 'D' :		
675		case 'U' :		
676		case 'Z' :		
677		case 'W' :		
678		return 1 ;		
679		default : return 0 ;		
680		}		
681		} /* is_drive */		
682		# endif /* USE_MACROS */		
683		# ifdef USE_MACROS		
684		# define map_symbol(c)((c)=='D'?LM_LOGIC_ZERO:(c)=='U'?LM_LOGIC_ONE:(c)=='Z'?LM_LOGIC_FLOAT:(c)=='W'?LM_LOGIC_UNKNOWN:LM_LOGIC_SIMULATOR)		
685		# else /* USE_MACROS */		
686		static long		
687		map_symbol (c)		
688		char c ;		
689		{ switch (c)		
690		{ case 'D' : return LM_LOGIC_ZERO ;		
691		case 'U' : return LM_LOGIC_ONE ;		
692		case 'Z' : return LM_LOGIC_FLOAT ;		
693		case 'W' : return LM_LOGIC_UNKNOWN ;		
694		default : return LM_LOGIC_SIMULATOR ;		
695		}		
696		} /* map_symbol */		
697		# endif /* USE_MACROS */		
698		# ifdef USE_MACROS		
699		# define unmap_symbol(lev)((lev)==LM_LOGIC_ZERO (lev)==LM_LOGIC_SOFT_ZERO?'L':(lev)==LM_LOGIC_ONE (lev)==LM_LOGIC_SOFT_ONE?'H':(lev)==LM_LOGIC_FLOAT?'T':(lev)==LM_LOGIC_UNKNOWN?'X':'?')		
700		# else /* USE_MACROS */		
701		static char		
702		unmap_symbol (lev)		
703		long lev ;		
704		{ switch (lev)		
705		{ case LM_LOGIC_ZERO : case LM_LOGIC_SOFT_ZERO : return 'L' ;		
706		case LM_LOGIC_ONE : case LM_LOGIC_SOFT_ONE : return 'H' ;		
707		case LM_LOGIC_FLOAT : return 'T' ;		
708		case LM_LOGIC_UNKNOWN : return 'X' ;		
709		default : return '?' ;		
710		}		
711		} /* unmap_symbol */		
712		# endif /* USE_MACROS */		
713		# ifdef USE_MACROS		
714				
715				
716				
717				
718				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 7/18
LINE #		SOURCE TEXT		
719		# Define match(val,sym)((val)--LM_LOGIC_SIMULATOR (sym)=='X'?1:(sym)=='L'?7:(val)--LM_LOGIC_ZERO (val)--LM_LOGIC_SOFT_ZERO:(sym)=='H'?7:(val)--LM_LOGIC_ONE (val)--LM_LOGIC_SOFT_ONE:(sym)=='T'?7:(val)--LM_LOGIC_FLOAT:0)		
720		# else /* USE_MACROS */		
721		static unsigned short		
722		match (val , sym)		
723		{		
724		if (val == LM_LOGIC_SIMULATOR sym == 'X')		
725		return 1;		
726		switch (sym)		
727		{		
728		case 'L': return val == LM_LOGIC_ZERO val == LM_LOGIC_SOFT_ZERO;		
729		case 'H': return val == LM_LOGIC_ONE val == LM_LOGIC_SOFT_ONE;		
730		case 'T': return val == LM_LOGIC_FLOAT;		
731		default: return 0;		
732		}		
733		/* match */		
734		#endif /* USE_MACROS */		
735		lm_play_test_vectors		
736		{		
737		instance_or_fault_id,		
738		test_vector_file,		
739		discrepancy_report_file,		
740		discrepancy_cutoff,		
741		number_of_sets,		
742		number_of_time_settings,		
743		number_of_pis_map_settings,		
744		number_of_discrepancies		
745		}		
746		long instance_or_fault_id,		
747		char		
748		{		
749		test_vector_file,		
750		discrepancy_report_file,		
751		unsigned long discrepancy_cutoff,		
752		unsigned long		
753		{		
754		number_of_sets,		
755		number_of_time_settings,		
756		number_of_pis_map_settings,		
757		number_of_discrepancies;		
758		{		
759		register long ret;		
760		long		
761		pis_id,		
762		value,		
763		dout_care,		
764		dout_care2,		
765		dout_care3,		
766		dout_care4,		
767		char		
768		{		
769		is_in,		
770		is_out,		
771		i or_o,		
772		done,		
773		mess,		
774		register unsigned long column;		
775		unsigned long		
776		max_column = 0;		
777		warnings = 0;		
778		register struct test_pis *test_pis,		
779		set_count = number_of_sets = 0;		
780		get_count = number_of_gets = 0;		
781		vector_count = number_of_time_settings = 0;		
782		pis_count = number_of_pis_map_settings = 0;		
783		disc_count = number_of_discrepancies = 0;		
784		while		
785		{		
786		(void) lm_get_message (&dout_care , &mess);		
787		dout_care != LM_NO_MORE_MESSAGES		
788		}		
789		tv_file = stdin;		
790		dr_file = stdout;		
791		if (test_vector_file != LM_NO_FILE)		
792		{		
793		ret =		
794		lm_resolve_library_file		
795		(LM_LIBRARY_VARIABLE_NAME ,		
796		test_vector_file ,		
797		tv_filename ,		
798		(long) sizeof (tv_filename)		
799);		
800		if (ret == LM_ERROR)		
801		return LM_ERROR;		
802		while		
803		{		
804		(void) lm_get_message (&dout_care , &mess);		
805		dout_care != LM_NO_MORE_MESSAGES		
806		{		
807		void) fprintf (stderr , "LM modeler warning: %s\n" , mess);		
808		if ((tv_file = fopen (tv_filename , "r")) == NULL)		
809		{		
810		lm_queue_message		
811		(ERROR_MSG ,		
812		"can't open test vector file: %s" ,		
813		tv_filename		
814);		
815		return LM_ERROR;		
816		}		
817		if (tv_file != stdin)		
818		{		
819		(void) fclose (tv_file);		
820		return LM_ERROR;		
821		}		
822		ret =		
823		lm_inquire_instance		
824		(instance_or_fault_id ,		
825		(long) LM_ASSOCIATED_DEFINITION_ID ,		
826		&definition_id		
827);		
828		if (ret == LM_ERROR)		
829		return LM_ERROR;		
830		while		
831		{		
832		(void) lm_get_message (&dout_care , &mess);		
833		dout_care != LM_NO_MORE_MESSAGES		
834		{		
835		(void) fprintf (stderr , "LM modeler warning: %s\n" , mess);		
836		do		
837		{		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89 TIME 1:20:52 pm	PAGE # 8/19
LINE #	SOURCE TEXT			
838	if (! hscan ())			
839	return LM_ERROR ;			
840	while ("tok == '\n'") ,			
841	if (strcmp (tok , "test"))			
842	return tv_error ("test expected") ,			
843	do			
844	if (! hscan ())			
845	return LM_ERROR ;			
846	while ("tok == '\n'") ,			
847	do			
848	{ if (! is_name (tok))			
849	return tv_error ("signal name expected") ,			
850	(void) strcpy (pin , tok) ,			
851	if (! hscan ())			
852	return LM_ERROR ;			
853	if (! is_column (tok))			
854	return tv_error ("column number expected after signal name") ,			
855	column = value_of (tok) ,			
856	if (column == 0 column > MAX_PIN_COUNT)			
857	return tv_error ("column number out of range") ,			
858	if (! hscan ())			
859	return LM_ERROR ;			
860	if ("tok == 'I' tok == 'O'")			
861	return tv_error ("I or O expected after column number") ,			
862	if ("tok == 'I'")			
863	{ is_in = 1 ;			
864	is_out = 0 ;			
865	}			
866	else			
867	{ is_in = 0 ;			
868	is_out = 1 ;			
869	}			
870	i_or_o = tok ;			
871	if (! hscan ())			
872	return LM_ERROR ;			
873	if ("tok == '('")			
874	{ if (! hscan ())			
875	return LM_ERROR ;			
876	if (! is_name (tok))			
877	return tv_error ("signal name expected after '('") ,			
878	if (! hscan ())			
879	return LM_ERROR ;			
880	if (! is_state_char (tok))			
881	return tv_error ("state character expected after signal name") ,			
882	do			
883	if (! hscan ())			
884	return LM_ERROR ;			
885	while (is_state_char (tok)) ,			
886	if ("tok == '('")			
887	return tv_error ("('" expected after state character(s)") ,			
888	if (! hscan ())			
889	return LM_ERROR ;			
890	if ("tok == 'I' tok == 'O'")			
891	{ if ("tok == i_or_o")			
892	return tv_error ("duplicate pin direction") ,			
893	if ("tok == 'I'")			
894	is_in = 1 ;			
895	else			
896	is_out = 1 ;			
897	if (! hscan ())			
898	return LM_ERROR ;			
899	if ("tok == '('")			
900	{ if (! hscan ())			
901	return LM_ERROR ;			
902	if (! is_name (tok))			
903	return tv_error ("signal name expected after '('") ,			
904	if (! hscan ())			
905	return LM_ERROR ;			
906	if (! is_state_char (tok))			
907	return			
908	tv_error			
909	("state character expected after signal name"			
910) ,			
911	do			
912	if (! hscan ())			
913	return LM_ERROR ;			
914	while (is_state_char (tok)) ,			
915	if ("tok == '('")			
916	/* These tokens must remain on a single line otherwise			
917	the Apollo SYSS compiler/preprocessor will die.			
918	*/			
919	return tv_error ("('" expected after state character(s)") ,			
920	if (! hscan ())			
921	return LM_ERROR ;			
922	if ("tok == '\n'")			
923	do			
924	if (! hscan ())			
925	return LM_ERROR ;			
926	while ("tok == '\n'") ,			
927	else			
928	return tv_error ("newline expected") ,			
929	}			
930	else if ("tok == '\n'")			
931	do			
932	if (! hscan ())			
933	return LM_ERROR ;			
934	while ("tok == '\n'") ,			
935	else			
936	return tv_error ("newline expected") ,			
937	}			
938	else if ("tok == '\n'")			
939	do			
940	if (! hscan ())			
941	return LM_ERROR ;			
942	while ("tok == '\n'") ,			
943	else			
944	return tv_error ("newline expected") ,			
945	}			
946	else if ("tok == 'I' tok == 'O'")			
947	{ if ("tok == i_or_o")			
948	return tv_error ("duplicate pin direction") ,			
949	if ("tok == 'I'")			
950	is_in = 1 ;			
951	else			
952	is_out = 1 ;			
953	if (! hscan ())			
954	return LM_ERROR ;			
955	if ("tok == '('")			
956	{ if (! hscan ())			
957	return LM_ERROR ;			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 9/20
SOURCE TEXT				
LINE #				
958	if (! is_name (tok))			
959	return tv_error ("signal name expected after '(' " ,			
960	if (! is_name (tok))			
961	return LM_ERROR ;			
962	if (! is_state_char (tok))			
963	return			
964	tv_error ("state character expected after signal name"			
965) ;			
966	do			
967	if (! is_name (tok))			
968	return LM_ERROR ;			
969	while (is_state_char (tok)) ,			
970	if ("tok != '")			
971	return tv_error ("' expected after state character(s)" ,			
972	return tv_error ("' expected after state character(s)" ,			
973	if (! is_name (tok))			
974	return LM_ERROR ;			
975	if ("tok == '\n'")			
976	do			
977	if (! is_name (tok))			
978	return LM_ERROR ;			
979	while ("tok == '\n'") ,			
980	else			
981	return tv_error ("newline expected") ;			
982	else if ("tok == '\n'")			
983	do			
984	if (! is_name (tok))			
985	return LM_ERROR ;			
986	while ("tok == '\n'") ,			
987	else			
988	return tv_error ("newline expected") ;			
989	else if ("tok == '\n'")			
990	do			
991	if (! is_name (tok))			
992	return LM_ERROR ;			
993	while ("tok == '\n'") ,			
994	else			
995	return tv_error ("newline expected") ;			
996	++pin_count ;			
997	if (column > max_column)			
998	max_column = column ;			
999	if (! is_character_defined_pinmap_flag)			
1000	{ ret =			
1001	lm_set_pin_map			
1002	{ definition_id ,			
1003	pin ,			
1004	{ long } IN_BOARD_CASE ,			
1005	{ long } --column			
1006) ;			
1007	if (ret == LM_ERROR)			
1008	return LM_ERROR ;			
1009	while			
1010	{ (void) lm_get_message (& dest_care , & mess) ;			
1011	dest_care != LM_NO_MORE_MESSAGES			
1012	{			
1013	(void) fprintf (stderr , "LM modeler warning: %s\n" , mess) ;			
1014	}			
1015	test_pin = test_pin + column ;			
1016	test_pin -> tv_sys = 'I' ;			
1017	test_pin -> is_in = is_in ;			
1018	test_pin -> is_out = is_out ;			
1019	test_pin -> is_val = LM_LOGIC_SIMULATOR ;			
1020	test_pin -> out_val = LM_LOGIC_SIMULATOR ;			
1021	while ("tok != '\0' & strump (tok , "patterns")) ,			
1022	if (pin_count != max_column)			
1023	{ lm_queue_message			
1024	{ ERROR_MSG ,			
1025	"discontinuous columns in %s" ,			
1026	"test vector file"			
1027	} ;			
1028	if ("tok == '\0'")			
1029	return tv_error ("premature EOF") ;			
1030	done = 0 ;			
1031	do			
1032	{ do			
1033	if (! is_name (tok))			
1034	return LM_ERROR ;			
1035	while ("tok == '\n'") ,			
1036	if ("tok == '\0'")			
1037	done = 1 ;			
1038	else			
1039	{ ++vector_count ;			
1040	if (! is_name (tok))			
1041	return tv_error ("vector timestamp expected") ;			
1042	timestamp = float_value (tok) ;			
1043	if (timestamp == -1)			
1044	return LM_ERROR ;			
1045	ret =			
1046	lm_set_simulation_time ((unsigned long) 0 , timestamp) ;			
1047	if (ret == LM_ERROR)			
1048	return LM_ERROR ;			
1049	while			
1050	{ (void) lm_get_message (& dest_care , & mess) ;			
1051	dest_care != LM_NO_MORE_MESSAGES			
1052	{			
1053	(void) fprintf (stderr , "LM modeler warning: %s\n" , mess) ;			
1054	}			
1055	column = 0 ;			
1056	do			
1057	{ if (! is_name (tok))			
1058	return LM_ERROR ;			
1059	if (! is_state_char (tok))			
1060	{ if (! is_name_direction ("tok"))			
1061	return tv_error ("unknown direction symbols illegal") ;			
1062	test_pin [column++] . tv_sys = tok ;			
1063	}			
1064	while (is_state_char (tok)) ,			
1065	if ("tok == '\n'")			
1066	return tv_error ("newline expected") ;			
1067	if (column != pin_count)			
1068	return tv_error ("wrong number of columns") ;			
1069	do			
1070	{ ret =			
1071	lm_get_pin_change			
1072	{ instance_or_fault_id ,			
1073	& dest_care ,			
1074	}			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/lmtest.c

DATE 5/23/89
TIME 1:20:52 pm
PAGE # 10/21

SOURCE TEXT

```

1078      & pin_id ,
1079      & value ,
1080      & dont_care3 ,
1081      ( unsigned long ) & dont_care3 ,
1082      ( unsigned long ) & dont_care4 ,
1083      ( unsigned long ) & dont_care5 ,
1084      ) ,
1085      if ( ret == LM_ERROR )
1086      return LM_ERROR ;
1087      while
1088      ( (void) lm_get_message ( & dont_care , & mess ) ,
1089        dont_care != LM_NO_MORE_MESSAGES
1090      )
1091      {
1092        void ) fprintf
1093        ( stderr ,
1094          "LM modular warning: %s\n" ,
1095          mess
1096        ) ;
1097        ++get_count ;
1098        if ( pin_id != LM_INVALID_PIN_ID )
1099        {
1100          if ( pin_id == pin_count )
1101          {
1102            return tv_internal ( "bad pin_id" ) ,
1103            test_pin [ pin_id ] . out_val = value ,
1104          }
1105        }
1106        while ( pin_id != LM_INVALID_PIN_ID ) ,
1107        for
1108        {
1109          test_pin = test_pin , column = 0 ,
1110          column < pin_count ,
1111          ++test_pin , ++column
1112        }
1113        if ( test_pin -> is_out )
1114        {
1115          if ( is_sense ( test_pin -> tv_sym ) )
1116          {
1117            if ( ! match ( test_pin -> out_val , test_pin -> tv_sym ) )
1118              break ;
1119          }
1120          else if ( test_pin -> tv_sym != 'T' )
1121          {
1122            if
1123            {
1124              test_pin -> out_val != LM_LOGIC_SOFT_ZERO &&
1125              test_pin -> out_val != LM_LOGIC_FLOAT &&
1126              test_pin -> out_val != LM_LOGIC_SOFT_ONE
1127            }
1128            break ;
1129          }
1130          else
1131          {
1132            if ( ! match ( test_pin -> out_val , 'T' ) )
1133              break ;
1134          }
1135          if ( column != pin_count )
1136          {
1137            ++disc_count ;
1138            ( void ) fprintf
1139            ( dr_file ,
1140              "discrepancy at vector %lu, time %lu:\n" ,
1141              vector_count ,
1142              timestamp
1143            ) ;
1144            for ( column = 0 , column < pin_count , ++column )
1145            {
1146              ( void ) fprintf
1147              ( dr_file ,
1148                "tc",
1149                test_pin [ column ] . tv_sym
1150              ) ;
1151              ( void ) fprintf ( dr_file , "\n" ) ;
1152            }
1153            for
1154            {
1155              test_pin = test_pin , column = 0 ,
1156              column < pin_count ,
1157              ++test_pin , ++column
1158            }
1159            {
1160              i_or_o = " ,
1161              if ( test_pin -> is_out )
1162              {
1163                if ( is_sense ( test_pin -> tv_sym ) )
1164                {
1165                  if
1166                  {
1167                    ! match
1168                    {
1169                      test_pin -> out_val ,
1170                      test_pin -> tv_sym
1171                    }
1172                    i_or_o = unmap_symbol ( test_pin -> out_val ) ,
1173                  }
1174                  else if ( test_pin -> tv_sym != 'T' )
1175                  {
1176                    if
1177                    {
1178                      test_pin -> out_val != LM_LOGIC_SOFT_ZERO &&
1179                      test_pin -> out_val != LM_LOGIC_FLOAT &&
1180                      test_pin -> out_val != LM_LOGIC_SOFT_ONE
1181                    }
1182                    i_or_o = unmap_symbol ( test_pin -> out_val ) ,
1183                  }
1184                  else
1185                  {
1186                    if ( ! match ( test_pin -> out_val , 'T' ) )
1187                      i_or_o = unmap_symbol ( test_pin -> out_val ) ,
1188                    ( void ) fprintf ( dr_file , "tc" , i_or_o ) ,
1189                  }
1190                  ( void ) fprintf ( dr_file , "\n" ) ;
1191                  if ( ! discrepancy_cutoff )
1192                    done = 1 ;
1193                }
1194              }
1195              if ( ! done )
1196              {
1197                for
1198                {
1199                  test_pin = test_pin , column = 0 ,
1200                  column < pin_count ,
1201                  ++test_pin , ++column
1202                }
1203                if ( is_drive ( test_pin -> tv_sym ) )
1204                {
1205                  if
1206                  {
1207                    value = map_symbol ( test_pin -> tv_sym ) !=
1208                    test_pin -> in_val
1209                  }
1210                  {
1211                    test_pin -> in_val = value ,
1212                    ret =
1213                    lm_get_pin_value
1214                    {
1215                      instance or fault_id ,
1216                      ( long ) LM_DONT_CARE ,
1217                      ( long ) column ,
1218                      value
1219                    }
1220                  }
1221                  if ( ret == LM_ERROR )
1222                    return LM_ERROR ;
1223                }
1224                while
1225                {
1226                  (void) lm_get_message ( & dont_care , & mess ) ,
1227                  dont_care != LM_NO_MORE_MESSAGES
1228                }
1229              }
1230            }

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/lmtest.c	DATE 5/23/89	PAGE # 11/22
LINE #		SOURCE TEXT		
1198		(void) fprintf		
1199		(stderr ,		
1200		"LM modeler : msg: %s\n" ,		
1201		mess		
1202) ,		
1203		++set_count ,		
1204		}		
1205		}		
1206		else		
1207		if		
1208		(test_pin -> is_in &&		
1209		test_pin -> is_val != LM_LOGIC_FLOAT		
1210)		
1211		{ test_pin -> is_val = LM_LOGIC_FLOAT ,		
1212		ret =		
1213		lm_set_pin_value		
1214		{ instance_or_fault_id ,		
1215		(long) LM_DONT_CARE ,		
1216		(long) column ,		
1217		LM_LOGIC_FLOAT		
1218		}		
1219		if (ret == LM_ERROR)		
1220		return LM_ERROR ,		
1221		while		
1222		{ (void) lm_get_message (& dont_care , & mess) ,		
1223		dont_care != LM_NO_MORE_MESSAGES		
1224		}		
1225		{ void) fprintf		
1226		(stderr ,		
1227		"LM modeler warning: %s\n" ,		
1228		mess		
1229) ,		
1230		++set_count ,		
1231		}		
1232		}		
1233		}		
1234		while (! done) ,		
1235		if (tv_file != stdin)		
1236		{ void) fclose (tv_file) ,		
1237		if (dr_file != stdout)		
1238		{ void) fclose (dr_file) ,		
1239		number_of_sets = set_count ,		
1240		number_of_gets = get_count ,		
1241		number_of_time_settings = vector_count ,		
1242		number_of_pin_map_settings = pin_count ,		
1243		number_of_discrepancies = disc_count ,		
1244		return warnings > LM_WARNING : LM_SUCCESS ,		
1245		} /* lm_play_test_vectors */		
1246				
1247		/* end of LM-1000 Test Vector Player */		
1248				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/network.c	DATE 5/23/89 TIME 1:20:53 pm	PAGE # 1/23
LINE #	SOURCE TEXT			
1	/* SCCS ID: network.c rev.3.1, 4/24/89 at 07:35:58 */			
2	#include <stdio.h>			
3	#include "connex.h"			
4	#include "message.h"			
5	#include "network.h"			
6	#include "lafi.h"			
7	#include "lm_sfi.h"			
8	#include "globals.h"			
9				
10				
11	lm_initconn(lm_box_name, skt)			
12	char *lm_box_name,			
13	short *skt,			
14	{			
15	CONNECTION *conn,			
16	u_char			
17	char *lm_malloc();			
18	char *lm_calloc();			
19	char			
20	char			
21	if (lm_is_modeler_alive(lm_box_name, &state) != LM_SUCCESS)			
22	return(FAILURE);			
23				
24	/* Find an empty slot in lm_table_of_conns */			
25	for (i = 0; lm_table_of_conns[i] != NULL; ++i)			
26				
27				
28	if (i == MAX_SYSTEM_TABLE_SIZE) {			
29	lm_queue_message(ERROR_MSG, "internal error: conn table full");			
30	return(FAILURE);			
31	}			
32	conn = (CONNECTION *)DCALLOC((unsigned)i, (unsigned)sizeof(CONNECTION));			
33	if (conn == (CONNECTION *) NULL) {			
34	lm_queue_message(ERROR_MSG, "out of memory on host");			
35	return(FAILURE);			
36	}			
37				
38	conn->name = (char *)MALLOC((unsigned)(strlen(lm_box_name) + 1));			
39	if (conn->name == NULL) {			
40	lm_queue_message(ERROR_MSG, "out of memory on host");			
41	return(FAILURE);			
42	}			
43				
44	(void)strcpy(conn->name, lm_box_name);			
45				
46	lm_table_of_conns[i] = conn;			
47				
48	*skt = i;			
49				
50	if (lm_init_connection_for_client(conn) == FAILURE)			
51	return(FAILURE);			
52	return(SUCCESS);			
53	}			
54				
55	lm_close_modeler(skt)			
56	short skt,			
57	{			
58	CONNECTION *conn,			
59	conn = lm_table_of_conns[skt];			
60				
61	lm_close_connection_for_client(conn);			
62				
63	DFREE((char *)conn->name);			
64	DFREE((char *)conn);			
65				
66	lm_table_of_conns[skt] = NULL;			
67				
68				
69				
70				

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM	DATE	PAGE #
	sfi/sfi1.c	5/23/89	1/24
		TIME	1:20:53 pm

LINE #	SOURCE TEXT
1	/* SFC ID: sfi1.c rev 1.2, 5/10/89 at 00:51:06 */
2	#include <stdio.h>
3	#include <ctype.h>
4	#include <unistd.h>
5	#include <sys/types.h>
6	#include <sys/socket.h>
7	#include <sys/time.h>
8	#include <sys/stat.h>
9	#include <sys/param.h>
10	#include <sys/mman.h>
11	#include <sys/wait.h>
12	#include <sys/resource.h>
13	#include <sys/errno.h>
14	#include <sys/signal.h>
15	#include <sys/unistd.h>
16	#include <sys/malloc.h>
17	extern char *lm_malloc();
18	extern char *lm_realloc();
19	extern void lm_free();
20	extern void lm_init_vars();
21	/* the only static variable allowed in this file: */
22	static u_char begin_session_called = FALSE;
23	
24	lm_begin_modeling_session(simulator_type, hostname, username)
25	{
26	u_long simulator_type;
27	char *hostname;
28	char *username;
29	{
30	u_short error_count;
31	u_short warning_count;
32	version_type sim_version;
33	version_type sfi_version;
34	version_type last_compatible_version;
35	if (begin_session_called == TRUE) {
36	lm_queue_message(ERROR_MSG, "lm_begin_modeling_session() called more than once");
37	return (LM_ERROR);
38	}
39	/* Version number is needed in simulator_type: */
40	sim_version = simulator_type;
41	sfi_version = LM_LOGIC_SIMULATOR;
42	last_compatible_version = LAST_COMPATIBLE_SFI_VERSION;
43	switch (sim_version & SIM_TYPE_MASK) {
44	case (LM_FAST_SIMULATOR & SIM_TYPE_MASK):
45	case (LM_LOGIC_SIMULATOR & SIM_TYPE_MASK):
46	break;
47	default:
48	lm_queue_message(ERROR_MSG, "internal simulator error: illegal simulator type");
49	return (LM_ERROR);
50	}
51	/* Check Version Number: */
52	if ((sim_version > sfi_version)
53	((sim_version < last_compatible_version)
54	((sim_version < last_compatible_version)
55	((sim_version < last_compatible_version)
56	((sim_version < last_compatible_version)
57	((sim_version < last_compatible_version)
58	((sim_version < last_compatible_version)
59	((sim_version < last_compatible_version)
60	((sim_version < last_compatible_version)
61	((sim_version < last_compatible_version)
62	((sim_version < last_compatible_version)
63	((sim_version < last_compatible_version)
64	((sim_version < last_compatible_version)
65	((sim_version < last_compatible_version)
66	((sim_version < last_compatible_version)
67	((sim_version < last_compatible_version)
68	((sim_version < last_compatible_version)
69	((sim_version < last_compatible_version)
70	((sim_version < last_compatible_version)
71	((sim_version < last_compatible_version)
72	((sim_version < last_compatible_version)
73	((sim_version < last_compatible_version)
74	((sim_version < last_compatible_version)
75	((sim_version < last_compatible_version)
76	((sim_version < last_compatible_version)
77	((sim_version < last_compatible_version)
78	((sim_version < last_compatible_version)
79	((sim_version < last_compatible_version)
80	((sim_version < last_compatible_version)
81	((sim_version < last_compatible_version)
82	((sim_version < last_compatible_version)
83	((sim_version < last_compatible_version)
84	((sim_version < last_compatible_version)
85	((sim_version < last_compatible_version)
86	((sim_version < last_compatible_version)
87	((sim_version < last_compatible_version)
88	((sim_version < last_compatible_version)
89	((sim_version < last_compatible_version)
90	((sim_version < last_compatible_version)
91	((sim_version < last_compatible_version)
92	((sim_version < last_compatible_version)
93	((sim_version < last_compatible_version)
94	((sim_version < last_compatible_version)
95	((sim_version < last_compatible_version)
96	((sim_version < last_compatible_version)
97	((sim_version < last_compatible_version)
98	((sim_version < last_compatible_version)
99	((sim_version < last_compatible_version)
100	((sim_version < last_compatible_version)
101	((sim_version < last_compatible_version)
102	((sim_version < last_compatible_version)
103	((sim_version < last_compatible_version)
104	((sim_version < last_compatible_version)
105	((sim_version < last_compatible_version)
106	((sim_version < last_compatible_version)
107	((sim_version < last_compatible_version)
108	((sim_version < last_compatible_version)
109	((sim_version < last_compatible_version)
110	((sim_version < last_compatible_version)
111	((sim_version < last_compatible_version)
112	((sim_version < last_compatible_version)
113	((sim_version < last_compatible_version)
114	((sim_version < last_compatible_version)
115	((sim_version < last_compatible_version)
116	((sim_version < last_compatible_version)
117	((sim_version < last_compatible_version)
118	((sim_version < last_compatible_version)

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 2/25
LINE #		SOURCE TEXT		
119		DPRINTF(("lm_end_modeling_session()\n"));		
120		}		
121		#endif		
122		clear_errors();		
123		if (!lm_sim_verified) {		
124		return (LM_SUCCESS);		
125		}		
126		/* Output the last vector */		
127		lm_simulation_time = (u_long) - 1;		
128		if (lm_vector_file_ptr != NULL) {		
129		lm_output_all_test_vector(lm_vector_file_ptr);		
130		}		
131		save_lm_network_timeout = lm_network_timeout;		
132		lm_network_timeout = 1000; /* 1 second */		
133		for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i) {		
134		if (lm_system_table[i] != NULL) {		
135		if (lm_system_table[i]->state != USED)		
136		continue;		
137		lm_choose_conn(lm_system_table[i]->fd);		
138		lm_put_int(ABORT_CMD);		
139		if (lm_end_put(lm_system_table[i]->fd) == FAILURE) {		
140		lm_choose_conn(lm_system_table[i]->fd);		
141		lm_put_int(ABORT_CMD);		
142		if (lm_end_put(lm_system_table[i]->fd) == FAILURE) {		
143		lm_queue_message(ERROR_MSG, "internal error: failed to abort user on modeler: %s",		
144		lm_system_table[i]->name);		
145		continue;		
146		}		
147		if (lm_get_int() != ABORT_ANS) {		
148		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
149		lm_system_table[i]->name);		
150		} else {		
151		/* (void) lm_dequeue_errors(); Lets not deque for now */		
152		}		
153		}		
154		}		
155		(void) lm_close_read_write_connection();		
156		lm_uninit(); /* Free host resources */		
157		lm_sim_verified = 0;		
158		begin_session_called = FALSE;		
159		lm_message_types(terror_count, twarning_count);		
160		lm_network_timeout = save_lm_network_timeout;		
161		if (error_count != 0)		
162		return (LM_ERROR);		
163		if (warning_count != 0)		
164		return (LM_WARNING);		
165		return (LM_SUCCESS);		
166		}		
167		/* =====		
168		lm_create_definition_f(filename, definition_id, modeler_name, device_name)		
169		char *filename,		
170		long *definition_id,		
171		char **modeler_name,		
172		char **device_name,		
173		{		
174		SYSTEM_INFO *system_ptr;		
175		DEFINITION_INFO *def;		
176		char *mdevice_name;		
177		char *mmodeler_name;		
178		char *mdevice;		
179		u_long s;		
180		u_long hsf_size;		
181		u_short fd;		
182		u_short device_usage;		
183		u_short error_count;		
184		u_short warning_count;		
185		char full_filename[MAX_STRING_LENGTH];		
186		{		
187		#ifdef DEBUG		
188		if (lm_debug[1]) {		
189		DPRINTF(("lm_create_definition_f(%s, %x, %x, %x)\n",		
190		filename, definition_id, modeler_name, device_name));		
191		}		
192		#endif		
193		*definition_id = -1;		
194		*modeler_name = NULL;		
195		*device_name = NULL;		
196		clear_errors();		
197		if (!lm_sim_verified) {		
198		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
199		return (LM_ERROR);		
200		}		
201		if (lm_gbl_time_scale_number == -1) {		
202		lm_queue_message(ERROR_MSG, "internal simulator error: lm_set_simulation_tick() has not yet been called");		
203		return (LM_ERROR);		
204		}		
205		if (lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)		
206		return (LM_ERROR);		
207		if (lm_resolve_library_file("LM_LIB", filename,		
208		full_filename, (long) MAX_STRING_LENGTH) == FAILURE) {		
209		return (LM_ERROR);		
210		}		
211		if (lm_check_file_type(full_filename) == FAILURE) {		
212		return (LM_ERROR);		
213		}		
214		/* BSD/ST55 compatibility change to hardcode 0. KDOWNY to equal 0. */		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfil.c	DATE 5/23/89	PAGE # 3/26
LINE #		SOURCE TEXT		
239		if ((fd = open(full_filename, 0)) == -1) {		
240		lm_queue_message(ERROR_MSG, "cannot open shell software file: %s",		
241		full_filename);		
242		return (LM_ERROR);		
243		}		
244		lm_thuf_ptr = lm_temp_buffer;		
245		buf_size = lm_max_thuf_addr - lm_temp_buffer - 1;		
246		while ((n = (u_long) read(fd, lm_thuf_ptr, (int) buf_size)) > 0) {		
247		lm_thuf_ptr += n;		
248		buf_size -= n;		
249		if (buf_size == 0) {		
250		if (lm_extnd_thuf() == FAILURE)		
251		return (LM_ERROR);		
252		buf_size = lm_max_thuf_addr - lm_thuf_ptr - 1;		
253		}		
254		lm_thuf_ptr++ = "\0";		
255		}		
256		if (close(fd) != 0) {		
257		lm_queue_message(ERROR_MSG, "cannot close file: %s",		
258		full_filename);		
259		return (LM_ERROR);		
260		}		
261		/* Run the HSM front-end processor (lexical_pass()). lexical_pass() returns		
262		NULL if there are lexical errors.		
263		*/		
264		lm_init_vars("HSM");		
265		outbuf = lm_lexical_pass(full_filename, lm_temp_buffer, &device_name,		
266		&modeler_name, &device_usage);		
267		/*		
268		if (outbuf == NULL) {		
269		/* error encountered when parsing the DARDDEF */		
270		return (LM_ERROR);		
271		}		
272		if (&device_name == NULL) {		
273		lm_queue_message(ERROR_MSG, "no device_name statement in shell software: %s",		
274		full_filename);		
275		/* Free the buffer allocated by HSM front-end processor */		
276		free(outbuf);		
277		return (LM_ERROR);		
278		}		
279		if ((lm_gbl_simulator_type & SIM_TYPE_MASK) == (LM_FAULT_SIMULATOR & SIM_TYPE_MASK))		
280		device_usage = 0; /* PUBLIC */		
281		/*		
282		if (lm_find_device(&device_name, device_usage, &system_ptr) == FAILURE) {		
283		/* Free the buffer allocated by HSM front-end processor */		
284		free(outbuf);		
285		return (LM_ERROR);		
286		}		
287		/*		
288		/* We found a system (pointed by system_ptr) which has the device we are		
289		looking for. Send the definition to that system.		
290		*/		
291		if (lm_send_def(system_ptr, &def,		
292		&device_name, outbuf, (short *) &definition_id) == SUCCESS) {		
293		/*		
294		(void) strcpy(lm_cdf_modeler_name, &def->system_ptr->name);		
295		(void) strcpy(lm_cdf_device_name, &def->device_name);		
296		/*		
297		*modeler_name = lm_cdf_modeler_name;		
298		*device_name = lm_cdf_device_name;		
299		/* Free the buffer allocated by HSM front-end processor */		
300		free(outbuf);		
301		/*		
302		#ifdef DEBUG		
303		if (lm_debug[1])		
304		lm_dump_definition(&def);		
305		#endif		
306		*definition_id = &def->def_id;		
307		/*		
308		#ifdef DEBUG		
309		if (lm_debug[2]) {		
310		PRINTF("lm_create_definition (%X, %d, %s, %s)\n",		
311		*definition_id, *modeler_name, *device_name);		
312		}		
313		#endif		
314		lm_message_types(&error_count, &warning_count);		
315		/*		
316		if (error_count != 0)		
317		return (LM_ERROR);		
318		/*		
319		if (warning_count != 0)		
320		return (LM_WARNING);		
321		/*		
322		return (LM_SUCCESS);		
323		/*		
324		} else {		
325		/* Free the buffer allocated by HSM front-end processor */		
326		free(outbuf);		
327		/*		
328		*definition_id = -1;		
329		/*		
330		return (LM_ERROR);		
331		/*		
332		}		
333		/*		
334		lm_create_definition_s(string, definition_id, modeler_name, device_name)		
335		/*		
336		char *string;		
337		long *definition_id;		
338		char *modeler_name;		
339		char *device_name;		
340		{		
341		SYSTEM_INFO *system_ptr;		
342		DEFINITION_INFO *def;		
343		char *str;		
344		char *device_name;		
345		char *modeler_name;		
346		char *outbuf;		
347		u_short device_usage;		
348		u_short error_count;		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfil.c	DATE 5/23/89 TIME 1:20:53 pm	PAGE # 4/27
SOURCE TEXT				
LINE #				
359	u_short warning_count,			
360	char			
361	c,			
362	#ifdef DEBUG			
363	if (lm_debug[1]) {			
364	PRINTF(("lm_create_definition_s(ts, Otxt, Otxt, Otxt)\n",			
365	string, definition_id, modeler_name, device_name));			
366	}			
367	#endif			
368	*definition_id = -1;			
369	*modeler_name = NULL;			
370	*device_name = NULL;			
371	clear_errors();			
372	if ((lm_sim_verified) {			
373	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
374	return (LM_ERROR);			
375	}			
376	if ((lm_gbl_time_scale_number == -1) {			
377	lm_queue_message(ERROR_MSG, "internal simulator error: lm_set_simulation_tick() has not yet been called");			
378	return (LM_ERROR);			
379	}			
380	if (lm_first_call == TRUE) {			
381	lm_tbuf_ptr = lm_tbuf_buffer;			
382	lm_first_call = FALSE;			
383	}			
384	if (string != LM_END_OF_TEXT) {			
385	str = string;			
386	if (lm_max_tbuf_addr - lm_tbuf_ptr - 1 < strlen(str) + 1)			
387	if (lm_extend_tbuf() == FAILURE)			
388	return (LM_ERROR);			
389	while (c = *str++)			
390	*lm_tbuf_ptr++ = c;			
391	return (LM_SUCCESS);			
392	}			
393	*lm_tbuf_ptr++ = '\0';			
394	lm_first_call = TRUE;			
395	}			
396	/* Run the BML front-end processor (lexical_pass()); lexical_pass() returns			
397	/* NULL if there are lexical errors.			
398	*/			
399	lm_init_vars("BML");			
400	outbuf = lm_lexical_pass("NULLNAME", lm_tbuf_buffer, &device_name,			
401	&modeler_name, &device_usage);			
402	}			
403	if (outbuf == NULL) {			
404	/* error encountered when parsing the BMLDEF */			
405	return (LM_ERROR);			
406	}			
407	if (xdevice_name == NULL) {			
408	lm_queue_message(ERROR_MSG, "no device_name statement in Shell Software: ts",			
409	"NULLNAME");			
410	/* Free the buffer allocated by BML front-end processor */			
411	free(outbuf);			
412	return (LM_ERROR);			
413	}			
414	if ((lm_gbl_simulator_type & SIM_TYPE_MASK) == (LM_FAULT_SIMULATOR & SIM_TYPE_MASK))			
415	device_usage = 0; /* PUBLIC */			
416	}			
417	if (lm_find_device(xdevice_name, device_usage,			
418	&modeler_name, &system_ptr) == FAILURE) {			
419	/* Free the buffer allocated by BML front-end processor */			
420	free(outbuf);			
421	return (LM_ERROR);			
422	}			
423	/* We found a system (pointed by system_ptr) which has the device we are			
424	/* looking for. Load the definition to that system.			
425	*/			
426	if (lm_send_def(system_ptr, &def,			
427	&device_name, outbuf, (short *) definition_id)) {			
428	}			
429	(void) strcpy(lm_crd_modeler_name, def->system_ptr->name);			
430	(void) strcpy(lm_crd_device_name, def->device_name);			
431	*modeler_name = lm_crd_modeler_name;			
432	*device_name = lm_crd_device_name;			
433	/* Free the buffer allocated by BML front-end processor */			
434	free(outbuf);			
435	}			
436	#ifdef DEBUG			
437	if (lm_debug[1])			
438	lm_dump_definition(def);			
439	#endif			
440	*definition_id = def->def_id;			
441	}			
442	#ifdef DEBUG			
443	if (lm_debug[2]) {			
444	PRINTF(("lm_create_definition_s(XI, id, ts, ts)\n",			
445	*definition_id, *modeler_name, *device_name));			
446	}			
447	#endif			
448	lm_message_types(&error_count, &warning_count);			
449	if (error_count != 0)			
450	return (LM_ERROR);			
451	if (warning_count != 0)			
452	return (LM_WARNING);			
453	return (LM_SUCCESS);			
454	} else {			
455	free(outbuf);			
456	*definition_id = -1;			
457	return (LM_ERROR);			
458	}			
459	}			
460	lm_create_instance(definition_id, instance_id, device_info_string)			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 5/28
LINE #		SOURCE TEXT		
479	long	definition_id,		
480	long	*instance_id,		
481	char	*device_info_string,		
482				
483		{		
484	INSTANCE_INFO	*inst_ptr,		
485	PIN_INFO	*pin,		
486	long	ret,		
487	char	*name,		
488	u_short	i,		
489	u_short	out_count,		
490	short	pin_no,		
491	u_short	box_inst_id,		
492				
493	#ifdef DEBUG			
494		{		
495		if (lm_debug[1]) {		
496		DPRINTF(("lm_create_instance(%d, %d, %s)\n",		
497		definition_id, instance_id, device_info_string));		
498		}		
499	#endif			
500		clear_errors();		
501				
502		if (!lm_pin_verified) {		
503		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
504		return (LM_ERROR);		
505		*instance_id = -1;		
506				
507		if (!lm_verify_name(device_info_string,		
508		"device_info_string", MAX_STRING_LENGTH) == FAILURE)		
509		return (LM_ERROR);		
510				
511		name = (char *) MALLOC((unsigned) (strlen(device_info_string) + 1));		
512		if (name == NULL) {		
513		lm_queue_message(ERROR_MSG, "out of memory on host");		
514		return (LM_ERROR);		
515		}		
516		(void) strcpy(name, device_info_string);		
517				
518		if ((definition_id != lm_last_definition_id_specified)		
519		(definition_id == -1)) {		
520		if ((definition_id < 0) (definition_id >= lm_def_id_avail)) {		
521		DPRINTF(name);		
522		lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",		
523		definition_id);		
524		return (LM_ERROR);		
525		}		
526		if (lm_def_id_table[definition_id] == NULL) {		
527		lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",		
528		definition_id);		
529		DPRINTF(name);		
530		return (LM_ERROR);		
531		}		
532		lm_last_definition_id_specified = definition_id;		
533		lm_definition_selected = lm_def_id_table[definition_id];		
534		}		
535		lm_choose_coas(lm_definition_selected->system_ptr->fd);		
536		lm_put_inst(CREATE_INSTANCE_CMD);		
537		lm_put_short(lm_definition_selected->box_def_id);		
538		}		
539		for (i = 0; device_info_string[i] != '\0'; ++i)		
540		lm_put_char(device_info_string[i]);		
541		lm_put_char("\0");		
542				
543		if (!lm_end_put(lm_definition_selected->system_ptr->fd)) {		
544		DPRINTF(name);		
545		return (LM_ERROR);		
546		}		
547		if (lm_get_inst() != CREATE_INSTANCE_ANS) {		
548		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
549		lm_definition_selected->system_ptr->name);		
550		DPRINTF(name);		
551		return (LM_ERROR);		
552		}		
553		if ((ret = lm_dequeue_errors()) == LM_ERROR) {		
554		DPRINTF(name);		
555		return (LM_ERROR);		
556		}		
557		box_inst_id = lm_get_short();		
558				
559		if (lm_new_instance_info(lm_definition_selected, &inst_ptr) == FAILURE) {		
560		/* Release the instance to the modeler. */		
561		lm_choose_coas(lm_definition_selected->system_ptr->fd);		
562		lm_put_inst(RELEASE_INSTANCE_CMD);		
563		lm_put_short(box_inst_id);		
564		}		
565		if (lm_end_put(lm_definition_selected->system_ptr->fd)) {		
566		if (lm_get_inst() != RELEASE_INSTANCE_ANS) {		
567		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
568		lm_definition_selected->system_ptr->name);		
569		else		
570		(void) lm_dequeue_errors();		
571		}		
572		DPRINTF(name);		
573		return (LM_ERROR);		
574		}		
575		inst_ptr->box_id = lm_definition_selected->system_ptr->fd;		
576		inst_ptr->box_inst_id = box_inst_id;		
577		inst_ptr->assoc_inst_id = -1;		
578		inst_ptr->assoc_box_inst_id = -1; /* This is an instance not a fault. */		
579		inst_ptr->def_id = lm_definition_selected->def_id;		
580		inst_ptr->assoc_val_or_store_pin = FALSE;		
581		inst_ptr->device_info_string = name;		
582				
583		*instance_id = inst_ptr->inst_id;		
584				
585		out_count = lm_get_short();		
586		for (i = 0; i < out_count; ++i) {		
587		pin_no = lm_get_short();		
588		pin = inst_ptr->pin_table[pin_no];		
589		pin->out_value = lm_get_char();		
590		}		
591				
592		#ifdef DEBUG		
593		{		
594		if (lm_debug[2]) {		
595		DPRINTF(("lm_create_instance(%d, %d, %s)\n", "instance_id));		
596		}		
597		}		
598		#endif		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 6/29
LINE #		SOURCE TEXT		
599		return (ret);		
600		}		
601		lm_create_fault(instance_id, fault_id)		
602		long instance_id,		
603		fault_id,		
604		{		
605		int ret;		
606		{		
607		if (lm_debug[1]) {		
608		DPRINTF(("lm_create_fault(%d, %d)\n", instance_id, fault_id));		
609		}		
610		#endif		
611		clear_errors();		
612		if (!lm_sim_verified) {		
613		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
614		return (LM_ERROR);		
615		if ((lm_gbl_simulator_type & SIM_TYPE_MASK) == (LM_LOGIC_SIMULATOR & SIM_TYPE_MASK)) {		
616		lm_queue_message(ERROR_MSG, "internal simulator error: Logic Simulator should not create fault");		
617		return (LM_ERROR);		
618		ret = lm_local_create_fault((long) -1, instance_id, fault_id);		
619		#endif		
620		if ((ret == LM_SUCCESS) && lm_debug[2]) {		
621		DPRINTF(("lm_create_fault(%d, %d)\n", fault_id));		
622		}		
623		#endif		
624		return (ret);		
625		}		
626		/*		
627		lm_release_definition(definition_id)		
628		long definition_id,		
629		{		
630		u_short i,		
631		u_short error_count,		
632		u_short warning_count;		
633		{		
634		if (lm_debug[1]) {		
635		DPRINTF(("lm_release_definition(%d)\n", definition_id));		
636		}		
637		#endif		
638		clear_errors();		
639		if (!lm_sim_verified) {		
640		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
641		return (LM_ERROR);		
642		if (definition_id == LM_ALL_DEFINITIONS) {		
643		for (i = 0; i < lm_def_id_table_size; ++i) {		
644		if (lm_def_id_table[i] != NULL) {		
645		if (lm_release_one_definition((long) i) == FAILURE)		
646		return (LM_ERROR);		
647		}		
648		} else {		
649		if (lm_release_one_definition((long) definition_id) == FAILURE)		
650		return (LM_ERROR);		
651		}		
652		lm_message_types(error_count, warning_count);		
653		if (error_count != 0)		
654		return (LM_ERROR);		
655		if (warning_count != 0)		
656		return (LM_WARNING);		
657		return (LM_SUCCESS);		
658		}		
659		lm_release_instance(instance_id)		
660		long instance_id,		
661		{		
662		INSTANCE_INFO *instance;		
663		INSTANCE_INFO *temp;		
664		FILE_INFO *file_ptr;		
665		long ret;		
666		u_long save_simulation_time;		
667		u_short i;		
668		{		
669		if (lm_debug[1]) {		
670		DPRINTF(("lm_release_instance(%d)\n", instance_id));		
671		}		
672		#endif		
673		clear_errors();		
674		if (!lm_sim_verified) {		
675		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
676		return (LM_ERROR);		
677		lm_last_instance_id_specified = -1;		
678		lm_instance_selected = NULL;		
679		ret = LM_SUCCESS;		
680		if (instance_id == LM_ALL_INSTANCES) {		
681		for (i = 0; i < lm_inst_id_table_size; ++i) {		
682		instance = lm_inst_id_table[i];		
683		if (BOGUS_INSTANCE(instance))		
684		continue;		
685		if (instance->assoc_box_inst_id == -1) {		
686		if (lm_release_instance((long) i) == LM_ERROR)		
687		return (LM_ERROR);		
688		}		
689		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 7/30
LINE #		SOURCE TEXT		
719		} else {		
720		if ((instance_id < 0)		
721		if (is_..._id >= lm_inst_id_table_size)) {		
722		lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: %d specified",		
723		instance_id);		
724		return (LM_ERROR);		
725		}		
726		instance = lm_inst_id_table(instance_id);		
727		if (BOCUS_INSTANCE(instance)) {		
728		lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: %d specified",		
729		instance_id);		
730		return (LM_ERROR);		
731		}		
732		if (instance->assoc_box_inst_id != -1) {		
733		lm_queue_message(ERROR_MSG, "internal simulator error: instance id: %d is a fault",		
734		instance_id);		
735		return (LM_ERROR);		
736		}		
737		/* Release all faults associated to this instance */		
738		for (i = 0; i < lm_inst_id_table_size; ++i) {		
739		temp = lm_inst_id_table[i];		
740		if (BOCUS_INSTANCE(temp))		
741		continue;		
742		if (temp->assoc_box_inst_id == instance->box_inst_id) {		
743		if (lm_release_fault((long) i) == LM_ERROR)		
744		return (LM_ERROR);		
745		}		
746		}		
747		}		
748		if (lm_search_key_id(lm_timing_file_ptr,		
749		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
750		lm_close_and_delete_file_key_id(&lm_timing_file_ptr,		
751		(u_short) instance->inst_id);		
752		instance->tm_file = NULL;		
753		}		
754		if (lm_search_key_id(lm_vector_file_ptr,		
755		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
756		lm_output_inst_vector(instance);		
757		lm_close_and_delete_file_key_id(&lm_vector_file_ptr,		
758		(u_short) instance->inst_id);		
759		instance->vector_file = NULL;		
760		}		
761		lm_choose_coord(instance->box_id);		
762		lm_put_inst(RELEASE_INSTANCE_CMD);		
763		lm_put_short(instance->box_inst_id);		
764		if (!lm_end_put(instance->box_id))		
765		return (LM_ERROR);		
766		}		
767		if (lm_get_inst() != RELEASE_INSTANCE_MSG) {		
768		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
769		lm_def_id_table(instance->def_id)->system_ptr->name);		
770		return (LM_ERROR);		
771		}		
772		if ((ret = lm_queue_errors()) == LM_ERROR)		
773		return (LM_ERROR);		
774		}		
775		lm_rls_instance_info(lm_def_id_table(instance->def_id), instance);		
776		return (ret);		
777		}		
778		}		
779		lm_release_fault(fault_id)		
780		{		
781		long fault_id;		
782		{		
783		INSTANCE_INFO		
784		FILE_INFO		
785		long		
786		u_long		
787		u_short		
788		i;		
789		#ifdef DEBUG		
790		if (lm_debug[1]) {		
791		DPRINTF("lm_release_fault(%d)\n", fault_id);		
792		}		
793		#endif		
794		{		
795		clear_errors();		
796		if (!lm_sim_verified) {		
797		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
798		return (LM_ERROR);		
799		}		
800		lm_last_instance_id_specified = -1;		
801		lm_instance_selected = NULL;		
802		ret = LM_SUCCESS;		
803		if (fault_id == LM_ALL_FAULTS) {		
804		for (i = 0; i < lm_inst_id_table_size; ++i) {		
805		fault = lm_inst_id_table[i];		
806		if (BOCUS_INSTANCE(fault))		
807		continue;		
808		if (fault->assoc_box_inst_id != -1) {		
809		if (lm_release_fault((long) i) == LM_ERROR)		
810		return (LM_ERROR);		
811		}		
812		}		
813		} else {		
814		if ((fault_id < 0)		
815		(fault_id >= lm_inst_id_table_size)) {		
816		lm_queue_message(ERROR_MSG, "internal simulator error: invalid fault id: %d specified",		
817		fault_id);		
818		return (LM_ERROR);		
819		}		
820		fault = lm_inst_id_table(fault_id);		
821		if (BOCUS_INSTANCE(fault)) {		
822		lm_queue_message(ERROR_MSG, "internal simulator error: invalid fault id: %d specified",		
823		fault_id);		
824		return (LM_ERROR);		
825		}		
826		if (fault->assoc_box_inst_id == -1) {		
827		}		
828		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 8/31
LINE #		SOURCE TEXT		
819		lm_queue_message(ERROR_MSG, "internal simulator error: fault id: %d is an instance",		
820		fault_id);		
821		return (LM_ERROR);		
822		}		
823		if (lm_search_key_id(lm_timing_file_ptr,		
824		(u_short) fault->inst_id, &file_ptr) == SUCCESS) {		
825		lm_close_and_delete_file_key_id(&lm_timing_file_ptr,		
826		(u_short) fault->inst_id);		
827		fault->tm_file = NULL;		
828		}		
829		if (lm_search_key_id(lm_vector_file_ptr,		
830		(u_short) fault->inst_id, &file_ptr) == SUCCESS) {		
831		lm_output_last_vector(fault);		
832		lm_close_and_delete_file_key_id(&lm_vector_file_ptr,		
833		(u_short) fault->inst_id);		
834		fault->vector_file = NULL;		
835		}		
836		lm_choose_conn(fault->box_id);		
837		lm_put_int(RELEASE_FAULT_CMD);		
838		lm_put_short(fault->box_inst_id);		
839		lm_put_short(fault->assoc_box_inst_id);		
840		}		
841		if (!lm_end_put(fault->box_id))		
842		return (LM_ERROR);		
843		}		
844		if (lm_get_int() != RELEASE_FAULT_ANS) {		
845		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %a",		
846		lm_def_id_table[fault->def_id]->system_ptr->name);		
847		return (LM_ERROR);		
848		}		
849		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
850		return (LM_ERROR);		
851		}		
852		lm_rls_instance_info(lm_def_id_table[fault->def_id], fault);		
853		}		
854		}		
855		return (ret);		
856		}		
857		}		
858		}		
859		}		
860		}		
861		}		
862		}		
863		}		
864		}		
865		}		
866		}		
867		}		
868		}		
869		}		
870		}		
871		}		
872		}		
873		}		
874		}		
875		}		
876		}		
877		}		
878		}		
879		}		
880		}		
881		lm_set_simulation_tick(number, units)		
882		unsigned long number;		
883		long units;		
884		{		
885		{		
886		#ifdef DEBUG		
887		if (lm_debug[1]) {		
888		DPRINTF(("lm_set_simulation_tick(%d, %d)\n", number, units));		
889		}		
890		#endif		
891		{		
892		clear_errors();		
893		}		
894		if (!lm_sim_verified) {		
895		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
896		return (LM_ERROR);		
897		}		
898		if (lm_gbl_time_scale_number != -1) {		
899		lm_queue_message(ERROR_MSG, "internal simulator error: lm_set_simulation_tick() called more than once");		
900		return (LM_ERROR);		
901		}		
902		switch (units) {		
903		case LM_TENTHSECONDS:		
904		case LM_PICOSECONDS:		
905		case LM_NANOSECONDS:		
906		case LM_MICROSECONDS:		
907		break;		
908		default:		
909		lm_queue_message(ERROR_MSG, "internal simulator error: illegal unit: %d for simulation tick",		
910		units);		
911		return (LM_ERROR);		
912		}		
913		lm_gbl_time_scale_units = units;		
914		}		
915		if (number == 0) {		
916		lm_queue_message(ERROR_MSG, "internal simulator error: number has to be greater than 0");		
917		return (LM_ERROR);		
918		}		
919		lm_gbl_time_scale_number = number;		
920		}		
921		if (lm_set_time_scale(lm_gbl_time_scale_number, lm_gbl_time_scale_units) == FAILURE)		
922		return (LM_ERROR);		
923		return (LM_SUCCESS);		
924		}		
925		}		
926		}		
927		}		
928		}		
929		}		
930		lm_set_simulation_time(htime, ltime)		
931		unsigned long htime;		
932		unsigned long ltime;		
933		{		
934		{		
935		#ifdef DEBUG		
936		if (lm_debug[1]) {		
937		DPRINTF(("lm_set_simulation_time(%d, %d)\n", htime, ltime));		
938		}		
939		#endif		
940		{		
941		clear_errors();		
942		}		
943		if (!lm_sim_verified) {		
944		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
945		return (LM_ERROR);		
946		}		
947		if (lm_gbl_time_scale_number == -1) {		
948		lm_queue_message(ERROR_MSG, "internal simulator error: lm_set_simulation_tick() has not yet been called");		
949		return (LM_ERROR);		
950		}		
951		if (htime != 0) {		
952		lm_queue_message(ERROR_MSG, "internal simulator error: only 32-bit simulation time is currently supported");		
953		return (LM_ERROR);		
954		}		
955		if (ltime < lm_simulation_time) {		
956		lm_queue_message(ERROR_MSG, "internal simulator error: simulation time decreased from %ld to %ld", lm_simulation_time, ltime);		
957		return (LM_ERROR);		
958		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 9/32
TIME 1:20:53 pm				
SOURCE TEXT				
LINE #				
559	lm_simulation_time = itime;			
560	if (lm_vector_file_ptr != NULL) {			
561	lm_output_all_test_vector(lm_vector_file_ptr);			
562	}			
563	return (LM_SUCCESS);			
564	}			
565	lm_set_logic_level_map(attribute, value)			
566	long attribute;			
567	long value;			
568	{			
569	#ifdef DEBUG			
570	if (lm_debug[1]) {			
571	PRINTF(("lm_set_logic_level_map(%d, %d)\n", attribute, value));			
572	}			
573	#endif			
574	clear_errors();			
575	if (!lm_sim_verified) {			
576	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
577	return (LM_ERROR);			
578	}			
579	switch (attribute) {			
580	case LM_LOGIC_ZERO:			
581	lm_user_to_lm_logic_level_map(0).user_value = value;			
582	lm_user_to_lm_logic_level_map(0).lm_value = LOGIC_0;			
583	if (lm_user_to_lm_logic_level_map(0).set == 0) {			
584	lm_user_to_lm_logic_level_map(0).set = 1;			
585	--lm_logic_level_map_count;			
586	}			
587	lm_to_user_logic_level_map(LOGIC_0) = value;			
588	break;			
589	case LM_LOGIC_SOFT_ZERO:			
590	lm_user_to_lm_logic_level_map(1).user_value = value;			
591	lm_user_to_lm_logic_level_map(1).lm_value = LOGIC_S0;			
592	if (lm_user_to_lm_logic_level_map(1).set == 0) {			
593	lm_user_to_lm_logic_level_map(1).set = 1;			
594	--lm_logic_level_map_count;			
595	}			
596	lm_to_user_logic_level_map(LOGIC_S0) = value;			
597	break;			
598	case LM_LOGIC_ONE:			
599	lm_user_to_lm_logic_level_map(2).user_value = value;			
600	lm_user_to_lm_logic_level_map(2).lm_value = LOGIC_1;			
601	if (lm_user_to_lm_logic_level_map(2).set == 0) {			
602	lm_user_to_lm_logic_level_map(2).set = 1;			
603	--lm_logic_level_map_count;			
604	}			
605	lm_to_user_logic_level_map(LOGIC_1) = value;			
606	break;			
607	case LM_LOGIC_SOFT ONE:			
608	lm_user_to_lm_logic_level_map(3).user_value = value;			
609	lm_user_to_lm_logic_level_map(3).lm_value = LOGIC_S1;			
610	if (lm_user_to_lm_logic_level_map(3).set == 0) {			
611	lm_user_to_lm_logic_level_map(3).set = 1;			
612	--lm_logic_level_map_count;			
613	}			
614	lm_to_user_logic_level_map(LOGIC_S1) = value;			
615	break;			
616	case LM_LOGIC_FLOAT:			
617	lm_user_to_lm_logic_level_map(4).user_value = value;			
618	lm_user_to_lm_logic_level_map(4).lm_value = LOGIC_F1;			
619	if (lm_user_to_lm_logic_level_map(4).set == 0) {			
620	lm_user_to_lm_logic_level_map(4).set = 1;			
621	--lm_logic_level_map_count;			
622	}			
623	lm_to_user_logic_level_map(LOGIC_F1) = value;			
624	break;			
625	case LM_LOGIC_UNKNOWN:			
626	lm_user_to_lm_logic_level_map(5).user_value = value;			
627	lm_user_to_lm_logic_level_map(5).lm_value = LOGIC_U;			
628	if (lm_user_to_lm_logic_level_map(5).set == 0) {			
629	lm_user_to_lm_logic_level_map(5).set = 1;			
630	--lm_logic_level_map_count;			
631	}			
632	lm_to_user_logic_level_map(LOGIC_U) = value;			
633	break;			
634	default:			
635	lm_queue_message(ERROR_MSG, "illegal attribute for function");			
636	return (LM_ERROR);			
637	}			
638	return (LM_SUCCESS);			
639	}			
640	lm_set_pin_map(definition_id, pin_name, pin_type, pin_id)			
641	long definition_id;			
642	char pin_name;			
643	long pin_type;			
644	long pin_id;			
645	{			
646	DAB_PID *dab_pid_ptr;			
647	u_long key;			
648	u_short dab_pid_index;			
649	}			
650	#ifdef DEBUG			
651	if (lm_debug[1]) {			
652	PRINTF(("lm_set_pin_map(%d, %s, %d, %d)\n",			
653	definition_id, pin_name, pin_type, pin_id));			
654	}			
655	#endif			
656	}			
657	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	S	DATE 5/23/89	PAGE # 10/33
				TIME 1:20:53 pm	

LINE #	SOURCE TEXT
1079	clear_errors();
1080	
1081	if (!lm_sim_verified) {
1082	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");
1083	return (LM_ERROR);
1084	}
1085	if ((definition_id != lm_last_definition_id_specified)
1086	(definition_id == -1)) {
1087	
1088	if ((definition_id < 0) (definition_id > lm_def_id_avail)) {
1089	lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",
1090	definition_id);
1091	return (LM_ERROR);
1092	}
1093	if (lm_def_id_table[definition_id] == NULL) {
1094	lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",
1095	definition_id);
1096	return (LM_ERROR);
1097	}
1098	lm_last_definition_id_specified = definition_id;
1099	lm_definition_selected = lm_def_id_table[definition_id];
1100	
1101	if ((pin_type < LM_INPUT) (pin_type > LM_DONT_CARE)) {
1102	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_type: %d specified",
1103	pin_type);
1104	return (LM_ERROR);
1105	}
1106	if (!lm_verify_name(pin_name, "pin_name", MAX_STRING_LENGTH) == FAILURE)
1107	return (LM_ERROR);
1108	
1109	pin_type = pin_type - LM_INPUT;
1110	
1111	if ((pin_id < 0) (pin_id > MAX_PIN_ID)) {
1112	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_id: %d specified",
1113	pin_id);
1114	return (LM_ERROR);
1115	
1116	key = (pin_type << PIN_TYPE_BIT_OFFSET) pin_id;
1117	if (lm_bin_search(lm_definition_selected->dab_pid_table,
1118	key, dab_pid_index) == TRUE) {
1119	lm_queue_message(ERROR_MSG, "internal simulator error: duplicate pin map specification for pin: %s of device_name: %s",
1120	pin_name,
1121	lm_definition_selected->device_name);
1122	
1123	return (LM_ERROR);
1124	
1125	if (lm_search_pin_name(lm_definition_selected->dab_pid_table,
1126	lm_definition_selected->dab_pid_table_size,
1127	pin_name, dab_pid_index) == FALSE) {
1128	lm_queue_message(ERROR_MSG, "internal simulator error: pin_name: %s of device_name: %s is not in Shell Software",
1129	pin_name,
1130	lm_definition_selected->device_name);
1131	return (LM_ERROR);
1132	
1133	dab_pid_ptr = lm_definition_selected->dab_pid_table[dab_pid_index];
1134	switch ((dab_pid_ptr->key) >> PIN_TYPE_BIT_OFFSET) & MAX_PIN_TYPE) {
1135	case INVALID_KEY - LM_INPUT:
1136	lm_definition_selected->table_validated = FALSE;
1137	dab_pid_ptr->key = key;
1138	break;
1139	case LM_INPUT - LM_INPUT:
1140	case LM_OUTPUT - LM_INPUT:
1141	case LM_IO - LM_INPUT:
1142	case LM_DONT_CARE - LM_INPUT:
1143	
1144	/*
1145	* Conflicting specification for the same pin name. The pin was
1146	* specified as either INPUT/OUTPUT/IO/DONT_CARE before, and now the
1147	* same pin is being specified as INPUT/OUTPUT/
1148	* IO/IO_AS_INPUT/IO_AS_OUTPUT/DONT_CARE/ again;
1149	*/
1150	lm_queue_message(ERROR_MSG, "internal simulator error: conflicting pin_type specification for pin: %s of device_name: %s",
1151	pin_name,
1152	lm_definition_selected->device_name);
1153	return (LM_ERROR);
1154	case LM_IO_AS_INPUT - LM_INPUT:
1155	if ((pin_type == LM_INPUT - LM_INPUT)
1156	(pin_type == LM_OUTPUT - LM_INPUT)
1157	(pin_type == LM_IO - LM_INPUT)
1158	(pin_type == LM_IO_AS_INPUT - LM_INPUT)
1159	(pin_type == LM_DONT_CARE - LM_INPUT)) {
1160	lm_queue_message(ERROR_MSG, "internal simulator error: conflicting pin_type specification for pin: %s of device_name: %s",
1161	pin_name,
1162	lm_definition_selected->device_name);
1163	return (LM_ERROR);
1164	}
1165	/* the same specified key must be LM_IO_AS_OUTPUT --> OK */
1166	lm_definition_selected->table_validated = FALSE;
1167	dab_pid_ptr->key = key;
1168	
1169	break;
1170	case LM_IO_AS_OUTPUT - LM_INPUT:
1171	if ((pin_type == LM_INPUT - LM_INPUT)
1172	(pin_type == LM_OUTPUT - LM_INPUT)
1173	(pin_type == LM_IO - LM_INPUT)
1174	(pin_type == LM_IO_AS_OUTPUT - LM_INPUT)
1175	(pin_type == LM_DONT_CARE - LM_INPUT)) {
1176	lm_queue_message(ERROR_MSG, "internal simulator error: conflicting pin_type specification for pin: %s of device_name: %s",
1177	pin_name,
1178	lm_definition_selected->device_name);
1179	return (LM_ERROR);
1180	
1181	/* else the key must be LM_IO_AS_INPUT --> there is nothing to be done */
1182	break;
1183	default:
1184	break;
1185	}
1186	
1187	if (lm_insert_key(lm_definition_selected, key, (u_short) dab_pid_index) == FAILURE)
1188	return (LM_ERROR);
1189	
1190	return (LM_SUCCESS);
1191	}
1192	
1193	lm_set_pin_value(instance_or_fault_id, pin_type, pin_id, value)
1194	long
1195	instance_or_fault_id,
1196	pin_type,
1197	pin_id,
1198	value;
1199	{

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89 TIME 1:20:53 pm	PAGE # 11/34
LINE #	SOURCE TEXT			
1192	DEFINITION_INFO			
1193	-definition;			
1194	-dab_pid_ptr;			
1195	-pin_info;			
1196	-key;			
1197	-dab_pid;			
1198	-cur_index;			
1199	-prev_index;			
1200	-i;			
1201	-p_type;			
1202	-error_count;			
1203	-warning_count;			
1204	-pin_type_str[12];			
1205	-new_value;			
1206	-type_of_pin_change = NO_CNG;			
1207	-tmp_string;			
1208	#ifdef DEBUG			
1209	if (lm_debug[1]) {			
1210	DPRINTF(("lm_set_pin_value(td, td, td, td)\n",			
1211	instance_or_fault_id, pin_type, pin_id, value));			
1212	}			
1213	#endif			
1214	clear_errors();			
1215	if (!lm_sim_verified) {			
1216	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1217	return (LM_ERROR);			
1218	}			
1219	if (lm_logic_level_map_count != 0) {			
1220	if (lm_logic_level_map_count == 1) {			
1221	if (!lm_validate_logic_level_map()) == FAILURE)			
1222	return (LM_ERROR);			
1223	}			
1224	else {			
1225	lm_queue_message(ERROR_MSG, "internal simulator error: incomplete/missing logic_level_map specification");			
1226	return (LM_ERROR);			
1227	}			
1228	if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {			
1229	return (LM_ERROR);			
1230	}			
1231	/*			
1232	* Validate the definition if necessary. We only need to do this for			
1233	* instance and NOT for PART because we cannot create PART if we fail to			
1234	* create instance.			
1235	*/			
1236	definition = lm_def_id_table[lm_instance_selected->def_id];			
1237	if (definition->table_validated == FALSE) {			
1238	if (!lm_validate_definition(definition) == FAILURE)			
1239	return (LM_ERROR);			
1240	}			
1241	if ((pin_id < 0) (pin_id > MAX_PIN_ID)) {			
1242	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_id: td specified",			
1243	pin_id);			
1244	return (LM_ERROR);			
1245	}			
1246	if ((pin_type < LM_INPUT) (pin_type > LM_DONT_CARE)) {			
1247	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_type: td specified",			
1248	pin_type);			
1249	return (LM_ERROR);			
1250	}			
1251	p_type = pin_type - LM_INPUT;			
1252	key = (p_type << PIN_TYPE_BIT_OFFSET) pin_id;			
1253	if (lm_bin_search(definition->user_pid_hash_table, key, &dab_pid) == FALSE) {			
1254	lm_get_pin_type_str(pin_type_str, p_type);			
1255	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin (pin_type: %s pin_id: %d) of instance: %s specified",			
1256	pin_type_str, pin_id,			
1257	lm_instance_selected->device_info_string);			
1258	return (LM_ERROR);			
1259	}			
1260	pin = lm_instance_selected->pin_table[dab_pid];			
1261	if (pin == NULL) {			
1262	lm_queue_message(ERROR_MSG, "internal error: pin info missing");			
1263	return (LM_ERROR);			
1264	}			
1265	for (i = 0; i < MAX_LOGIC_STATE; ++i) {			
1266	if (value == lm_user_to_lm_logic_level_map[i].user_value)			
1267	break;			
1268	}			
1269	if (i == MAX_LOGIC_STATE) {			
1270	lm_queue_message(ERROR_MSG, "internal simulator error: illegal logic value: td specified",			
1271	value);			
1272	return (LM_ERROR);			
1273	}			
1274	dab_pid_ptr = definition->dab_pid_table[dab_pid];			
1275	new_value = lm_user_to_lm_logic_level_map[i].lm_value;			
1276	pin->sim_time = lm_simulation_time;			
1277	switch (dab_pid_ptr->pin_type) {			
1278	case IN:			
1279	/*			
1280	* set the out_value = in_value so we can return its value properly			
1281	* when the user calls lm_get_pin_value_2()			
1282	*/			
1283	pin->out_value = new_value;			
1284	case IO:			
1285	switch (dab_pid_ptr->pin_class) {			
1286	case DATA:			
1287	if (pin->is_linked == TRUE) {			
1288	if ((new_value != pin->in_value)			
1289	(pin->just_went_float == TRUE))			
1290	type_of_pin_change = DATA_CNG;			
1291	pin->in_value = new_value;			
1292	}			
1293	else {			
1294	if ((new_value != pin->in_value)			
1295	(pin->just_went_float == TRUE)) {			
1296	type_of_pin_change = DATA_CNG;			
1297	pin->is_linked = TRUE;			
1298	pin->in_value = new_value;			
1299	++lm_instance_selected->data_pin_change_count;			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 12/35
LINE #		SOURCE TEXT		
1319		pin->input_change = lm_instance_selected->data_pin_change_list;		
1320		lm_instance_selected->data_pin_change_list = dab_pid;		
1321		}		
1322		break;		
1323		case EVAL:		
1324		if (pin->is_linked == TRUE) {		
1325		if ((new_value != pin->in_value)		
1326		(pin->just_went_float == TRUE)) {		
1327		type_of_pin_change = EVAL_OR_STORE_CNG;		
1328		pin->in_value = new_value;		
1329		lm_queue_message(WARNING_MSG, "internal simulator warning: multiple changes on eval pin: %s of instance: %s",		
1330		definition->dab_pid_table[dab_pid].name,		
1331		lm_instance_selected->device_info_string);		
1332		}		
1333		} else {		
1334		if ((new_value != pin->in_value)		
1335		(pin->just_went_float == TRUE)) {		
1336		type_of_pin_change = EVAL_OR_STORE_CNG;		
1337		lm_instance_selected->seen_eval_or_store_pin = TRUE;		
1338		pin->is_linked = TRUE;		
1339		pin->in_value = new_value;		
1340		++lm_instance_selected->eval_pin_change_count;		
1341		pin->input_change = lm_instance_selected->eval_pin_change_list;		
1342		lm_instance_selected->eval_pin_change_list = dab_pid;		
1343		}		
1344		}		
1345		break;		
1346		case STORE:		
1347		case EDGE_RISE:		
1348		case EDGE_FALL:		
1349		if (pin->is_linked == TRUE) {		
1350		if ((new_value != pin->in_value)		
1351		(pin->just_went_float == TRUE)) {		
1352		type_of_pin_change = EVAL_OR_STORE_CNG;		
1353		pin->in_value = new_value;		
1354		lm_queue_message(WARNING_MSG, "internal simulator warning: multiple changes on store pin: %s of instance: %s",		
1355		definition->dab_pid_table[dab_pid].name,		
1356		lm_instance_selected->device_info_string);		
1357		}		
1358		} else {		
1359		if ((new_value != pin->in_value)		
1360		(pin->just_went_float == TRUE)) {		
1361		type_of_pin_change = EVAL_OR_STORE_CNG;		
1362		lm_instance_selected->seen_eval_or_store_pin = TRUE;		
1363		pin->is_linked = TRUE;		
1364		pin->in_value = new_value;		
1365		++lm_instance_selected->store_pin_change_count;		
1366		if (lm_instance_selected->store_pin_change_list == -1) {		
1367		/* First one on the list */		
1368		pin->input_change = lm_instance_selected->store_pin_change_list;		
1369		lm_instance_selected->store_pin_change_list = dab_pid;		
1370		} else {		
1371		cur_index = lm_instance_selected->store_pin_change_list;		
1372		prev_index = -1;		
1373		while ((cur_index != -1) && (dab_pid > cur_index)) {		
1374		prev_index = cur_index;		
1375		cur_index = lm_instance_selected->pin_table[cur_index]->		
1376		input_change;		
1377		}		
1378		if (prev_index == -1) {		
1379		/* Insert at head of list */		
1380		pin->input_change = lm_instance_selected->		
1381		store_pin_change_list;		
1382		lm_instance_selected->store_pin_change_list = dab_pid;		
1383		} else {		
1384		pin->input_change = lm_instance_selected->		
1385		pin_table[prev_index]->input_change;		
1386		lm_instance_selected->		
1387		pin_table[prev_index]->input_change = dab_pid;		
1388		}		
1389		}		
1390		break;		
1391		default:		
1392		break;		
1393		pin->just_went_float = FALSE;		
1394		break;		
1395		case OUT:		
1396		default:		
1397		switch (dab_pid_ptr->pin_type) {		
1398		case OUT:		
1399		tmp_string = "output";		
1400		break;		
1401		case POWER:		
1402		tmp_string = "power";		
1403		break;		
1404		case GROUND:		
1405		tmp_string = "ground";		
1406		break;		
1407		case NC:		
1408		tmp_string = "no connect";		
1409		break;		
1410		case NONE:		
1411		default:		
1412		tmp_string = "UNKNOWN";		
1413		break;		
1414		lm_queue_message(WARNING_MSG, "internal simulator warning: cannot set value on %s pin: %s of instance: %s",		
1415		tmp_string,		
1416		definition->dab_pid_table[dab_pid].name,		
1417		lm_instance_selected->device_info_string);		
1418		break;		
1419		}		
1420		if (lm_instance_selected->vector_file != NULL) {		
1421		switch (type_of_pin_change) {		
1422		case DATA_CNG:		
1423		case EVAL_OR_STORE_CNG:		
1424		++lm_instance_selected->pin_change_count;		
1425		lm_instance_selected->last_pin_change_time = lm_simulation_time;		
1426		lm_instance_selected->type_of_pin_change = DATA_CNG;		
1427		if (lm_set_vector_buffer(lm_instance_selected, dab_pid) == FAILURE)		
1428		}		
1429		}		
1430		}		
1431		}		
1432		}		
1433		}		
1434		}		
1435		}		
1436		}		
1437		}		
1438		}		

SOURCE PROGRAM		DATE	PAGE #
sfi/sfi1.c		5/23/89	13/36
TIME		1:20:53 pm	
SOURCE TEXT			
LINE #	SOURCE TEXT		
439	return (LM_ERROR);		
440	break;		
441	case NO_CFG:		
442	/*		
443	*/		
444	}		
445	lm_message_types(&error_count, &warning_count);		
446	if (error_count != 0)		
447	return (LM_ERROR);		
448	if (warning_count != 0)		
449	return (LM_WARNING);		
450	return (LM_SUCCESS);		
451	}		
452	}		
453	}		
454	}		
455	}		
456	/*		
457	*/		
458	lm_get_pin_change(instance_or_fault_id, pin_type, pin_id, value,		
459	delay_type, min, typ, max)		
460	long instance_or_fault_id;		
461	long pin_type;		
462	long pin_id;		
463	long value;		
464	long delay_type;		
465	long min;		
466	long typ;		
467	long max;		
468	{		
469	PIN_INFO pin;		
470	DEFINITION_INFO def;		
471	u_long key;		
472	short dab_pid;		
473	u_short error_count;		
474	u_short warning_count;		
475	{		
476	if (lm_debug[1]) {		
477	fprintf(stderr, "lm_get_pin_change: %d, %d, %d, %d, %d, %d, %d, %d\n",		
478	instance_or_fault_id, pin_type, pin_id,		
479	value, delay_type, min, typ, max);		
480	}		
481	#endif		
482	clear_errors();		
483	if (!lm_sim_verified) {		
484	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
485	return (LM_ERROR);		
486	}		
487	if (lm_get_pin_value_type != GET_PIN_VALUE_1) {		
488	if (lm_get_pin_value_type == -1) {		
489	/*		
490	This is the first time this function is called because		
491	lm_get_pin_value_type is still -1.		
492	*/		
493	lm_get_pin_value_type = GET_PIN_VALUE_1;		
494	} else {		
495	lm_queue_message(ERROR_MSG, "internal simulator error: cannot mix lm_get_pin_change() and lm_get_pin_value_2()");		
496	return (LM_ERROR);		
497	}		
498	if (lm_logic_level_map_count != 0) {		
499	if (lm_logic_level_map_count == 1) {		
500	if (lm_verify_logic_level_map() == FAILURE)		
501	return (LM_ERROR);		
502	} else {		
503	lm_queue_message(ERROR_MSG, "internal simulator error: incomplete/missing logic_level_map specification");		
504	return (LM_ERROR);		
505	}		
506	if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
507	return (LM_ERROR);		
508	}		
509	/*		
510	Validate the definition if necessary. We only need to do this for		
511	INSTANCE and NOT for FAULT because we cannot create FAULT if we fail to		
512	create instance.		
513	*/		
514	definition = lm_def_id_table[lm_instance_selected->def_id];		
515	if (definition->table_validated == FALSE) {		
516	if (lm_validate_definition(definition) == FAILURE)		
517	return (LM_ERROR);		
518	}		
519	if (lm_instance_selected->eval_or_store_pin) {		
520	if (lm_instance_selected->output_change_list != -1) {		
521	/*		
522	The instance needs to be evaluated while there are still leftover		
523	output changes which have not been returned to the user.		
524	*/		
525	lm_queue_message(ERROR_MSG, "internal simulator error: instance/fault id: %d was evaluated while there were leftover output changes",		
526	instance_or_fault_id);		
527	(void) lm_eval_instance(lm_instance_selected);		
528	return (LM_ERROR);		
529	} else {		
530	if (lm_eval_instance(lm_instance_selected) == FAILURE) {		
531	return (LM_ERROR);		
532	}		
533	}		
534	if (lm_instance_selected->get_pin_change_called == FALSE) {		
535	lm_link_all_output_pins(lm_instance_selected);		
536	lm_instance_selected->get_pin_change_called = TRUE;		
537	}		
538	dab_pid = lm_instance_selected->output_change_list;		
539	if (dab_pid != -1) {		
540	pin = lm_instance_selected->pin_table[dab_pid];		
541	if (pin == NULL) {		
542	lm_queue_message(ERROR_MSG, "internal error: pin info missing");		
543	return (LM_ERROR);		
544	}		
545	lm_instance_selected->output_change_list = pin->output_change;		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 14/37
LINE #		SOURCE TEXT		
1559		*value = lm_to_user_logic_level_map(pin->out_value);		
1560		*... = definition->dab_pid_table(dab_pid).key;		
1561		*pin_id = key & MAX_PIN_ID;		
1562		*pin_type = ((key >> PIN_TYPE_BIT_OFFSET) & MAX_PIN_TYPE) + LM_INPUT;		
1563		*delay_type = pin->delay_type;		
1564		*min = pin->min_delay;		
1565		*typ = pin->typ_delay;		
1566		*max = pin->max_delay;		
1567		#ifdef DEBUG		
1568		if (lm_debug[2]) {		
1569		DPRINTF(("lm_get_pin_change(XX, %d, %d, %d, %d, %d, %d, %d)\n",		
1570		pin_type, pin_id, *value, *delay_type, *min, *typ, *max));		
1571		}		
1572		#endif		
1573		lm_message_types(&error_count, &warning_count);		
1574		if (error_count != 0)		
1575		return (LM_ERROR);		
1576		if (warning_count != 0)		
1577		return (LM_WARNING);		
1578		return (LM_SUCCESS);		
1579		}		
1580		*pin_type = -1;		
1581		*pin_id = LM_INVALID_PIN_ID;		
1582		*value = lm_to_user_logic_level_map(LOGIC_U);		
1583		*delay_type = LM_UNSPECIFIED_DELAY;		
1584		*min = 0;		
1585		*typ = 0;		
1586		*max = 0;		
1587		#ifdef DEBUG		
1588		if (lm_debug[2]) {		
1589		DPRINTF(("lm_get_pin_change(XX, %d, %d, %d, %d, %d, %d, %d)\n",		
1590		pin_type, pin_id, *value, *delay_type, *min, *typ, *max));		
1591		}		
1592		#endif		
1593		lm_message_types(&error_count, &warning_count);		
1594		if (error_count != 0)		
1595		return (LM_ERROR);		
1596		if (warning_count != 0)		
1597		return (LM_WARNING);		
1598		return (LM_SUCCESS);		
1599		}		
1600		lm_get_pin_value_2(instance_or_fault_id, pin_type, pin_id, value,		
1601		delay_type, min, typ, max)		
1602		long instance_or_fault_id;		
1603		long pin_type;		
1604		long pin_id;		
1605		long *value;		
1606		long *delay_type;		
1607		unsigned long *min;		
1608		unsigned long *typ;		
1609		unsigned long *max;		
1610		{		
1611		PIN INFO		
1612		DEFINITION INFO		
1613		u_long key;		
1614		u_short dab_pid;		
1615		short pinno;		
1616		u_short error_count;		
1617		u_short warning_count;		
1618		char pin_type_str[12];		
1619		#ifdef DEBUG		
1620		if (lm_debug[1]) {		
1621		DPRINTF(("lm_get_pin_value_2(%d, %d, %d, %d, %d, %d, %d, %d)\n",		
1622		instance_or_fault_id, pin_type, pin_id, value,		
1623		delay_type, min, typ, max));		
1624		}		
1625		#endif		
1626		clear_errors();		
1627		if (!lm_pin_verified) {		
1628		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1629		return (LM_ERROR);		
1630		if (lm_get_pin_value_type != GET_PIN_VALUE_2) {		
1631		if (lm_get_pin_value_type == -1) {		
1632		/*		
1633		* This is the first time this function is called because		
1634		* lm_get_pin_value_type is still -1.		
1635		*/		
1636		lm_get_pin_value_type = GET_PIN_VALUE_2;		
1637		} else {		
1638		lm_queue_message(ERROR_MSG, "internal simulator error: cannot mix lm_get_pin_change() and lm_get_pin_value_2()");		
1639		return (LM_ERROR);		
1640		}		
1641		if (lm_logic_level_map_count != 0) {		
1642		if (lm_logic_level_map_count == 1) {		
1643		if (lm_verify_logic_level_map() == FAILURE)		
1644		return (LM_ERROR);		
1645		} else {		
1646		lm_queue_message(ERROR_MSG, "internal simulator error: incomplete/missing logic_level_map specification");		
1647		return (LM_ERROR);		
1648		}		
1649		if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
1650		return (LM_ERROR);		
1651		}		
1652		/*		
1653		* validate the definition if necessary. We only need to do this for		
1654		* INSTANCE and NOT for FAULT because we cannot create FAULT if we fail to		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89 TIME 1:20:53 pm	PAGE # 15/38
SOURCE TEXT				
LINE #	SOURCE TEXT			
1679	/* create instance.			
1680	*/			
1681	definition = lm_def_id_table(lm_instance_selected->def_id);			
1682	if (definition->table_validated == FALSE) {			
1683	if (lm_validate_definition(definition) == FAILURE)			
1684	return (LM_ERROR);			
1685	}			
1686	if ((pin_id < 0) (pin_id > MAX_PIN_ID)) {			
1687	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_id: %d specified",			
1688	pin_id);			
1689	return (LM_ERROR);			
1690	}			
1691	if ((pin_type < LM_INPUT) (pin_type > LM_DONT_CARE)) {			
1692	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_type: %d specified",			
1693	pin_type);			
1694	return (LM_ERROR);			
1695	}			
1696	pin_type = pin_type - LM_INPUT;			
1697	key = (pin_type << PIN_TYPE_BIT_OFFSET) pin_id;			
1698	if (lm_pin_search(definition->user_pid_hash_table, key, &dab_pid) == FALSE) {			
1699	lm_get_pin_type_str(pin_type_str, (u_char) pin_type);			
1700	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin (pin_type: %s pin_id: %d) of instance: %s specified",			
1701	pin_type_str, pin_id,			
1702	lm_instance_selected->device_info_string);			
1703	return (LM_ERROR);			
1704	}			
1705	if (lm_instance_selected->seen_eval_or_store_pin) {			
1706	if (lm_instance_selected->output_change_list != -1) {			
1707	/*			
1708	* The instance needs to be evaluated while there are still leftover			
1709	* output changes which have not been returned to the user.			
1710	lm_queue_message(ERROR_MSG, "internal simulator error: instance/fault id: %d was evaluated while there were leftover output changes",			
1711	instance_or_fault_id);			
1712	(void) lm_eval_instance(lm_instance_selected);			
1713	return (LM_ERROR);			
1714	} else {			
1715	if (lm_eval_instance(lm_instance_selected) == FALSE)			
1716	return (LM_ERROR);			
1717	}			
1718	pin = lm_instance_selected->pin_table[dab_pid];			
1719	if (pin == NULL) {			
1720	lm_queue_message(ERROR_MSG, "internal error: pin info missing");			
1721	return (LM_ERROR);			
1722	}			
1723	*value = lm_to_user_logic_level_map(pin->out_value);			
1724	/*			
1725	* delay_type = pin->delay_type;			
1726	*min = pin->min_delay;			
1727	*typ = pin->typ_delay;			
1728	*max = pin->max_delay;			
1729	*/			
1730	/*			
1731	* We assume that if the user calls lm_get_pin_value_2(), he will never call			
1732	* lm_get_pin_change(). So we can throw away the output pin change list.			
1733	*/			
1734	pinno = lm_instance_selected->output_change_list;			
1735	while (pinno != -1) {			
1736	pin = lm_instance_selected->pin_table[pinno];			
1737	pinno = pin->output_change;			
1738	pin->output_change = -1;			
1739	}			
1740	lm_instance_selected->output_change_list = -1;			
1741	/*			
1742	* if (lm_debug[2]) {			
1743	DPRINTF(("lm_get_pin_value_2(XX, XX, XX, %d, %d, %d, %d, %d)\n",			
1744	*value, "delay_type", "min", "typ", "max));			
1745	}			
1746	#endif			
1747	lm_message_types(&error_count, &warning_count);			
1748	if (error_count != 0)			
1749	return (LM_ERROR);			
1750	if (warning_count != 0)			
1751	return (LM_WARNING);			
1752	return (LM_SUCCESS);			
1753	}			
1754	/*			
1755	* lm_save_modeler_state(filename)			
1756	char *filename;			
1757	{			
1758	DEFINITION_INFO *def;			
1759	INSTANCE_INFO *instance;			
1760	INSTANCE_INFO *fault;			
1761	USER_PID *user_pid_ptr;			
1762	FILE_INFO *file_info_ptr;			
1763	long filesize;			
1764	u_short def_count;			
1765	u_short inst_count;			
1766	u_short i;			
1767	u_short user_pid_count;			
1768	u_short error_count;			
1769	u_short warning_count;			
1770	u_long unspc_delay_count;			
1771	u_char file_count;			
1772	unspc_delay_list_element *elem_ptr;			
1773	char c;			
1774	#endif			
1775	if (lm_debug[1]) {			
1776	DPRINTF(("lm_save_modeler_state(%s)\n", filename));			
1777	}			
1778	#endif			
1779	clear_errors();			
1780	if (!lm_sim_verified) {			
1781	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1782	return (LM_ERROR);			
1783	}			

Copyright: 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfil.c	DATE 5/23/89	PAGE # 16/39
LINE #		SOURCE TEXT		
1799		}		
1800		if (!lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)		
1801		return (LM_ERROR);		
1802		}		
1803		if ((lm_save_set = fopen(filename, "w")) == NULL) {		
1804		lm_queue_message(ERROR_MSG, "cannot open save-set file: %s for write",		
1805		filename);		
1806		return (LM_ERROR);		
1807		}		
1808		/* save the save-set version */		
1809		write_long(lm_save_set, SAVE_SET_VERSION_ID);		
1810		/* save dummy byte count */		
1811		write_long(lm_save_set, 0);		
1812		/* save simulation tick */		
1813		write_long(lm_save_set, lm_gbl_time_scale_units);		
1814		write_long(lm_save_set, lm_gbl_time_scale_number);		
1815		/* save the current simulation time */		
1816		write_long(lm_save_set, lm_simulation_time);		
1817		/* save the logic map */		
1818		for (i = 0; i < MAX_LOGIC_STATE; ++i) {		
1819		write_short(lm_save_set, lm_user_to_lm_logic_level_map[i].user_value);		
1820		write_short(lm_save_set, lm_user_to_lm_logic_level_map[i].lm_value);		
1821		}		
1822		/* save lm_get_pid_value_type */		
1823		write_short(lm_save_set, lm_get_pid_value_type);		
1824		/* save the Timing Measurement File information */		
1825		file_count = 0;		
1826		file_info_ptr = lm_timing_file_ptr;		
1827		while (file_info_ptr != NULL) {		
1828		++file_count;		
1829		file_info_ptr = file_info_ptr->next;		
1830		}		
1831		write_char(lm_save_set, file_count);		
1832		file_info_ptr = lm_timing_file_ptr;		
1833		while (file_info_ptr != NULL) {		
1834		write_short(lm_save_set, file_info_ptr->instance_id);		
1835		for (i = 0; (c = file_info_ptr->name[i]) != '\0'; ++i) {		
1836		write_char(lm_save_set, c);		
1837		}		
1838		write_char(lm_save_set, c);		
1839		if (lm_get_file_size(file_info_ptr->file_ptr,		
1840		file_info_ptr->name, &filesize) == FAILURE) {		
1841		filesize = 0;		
1842		write_long(lm_save_set, filesize);		
1843		file_info_ptr = file_info_ptr->next;		
1844		}		
1845		/* save the Test Vector File information */		
1846		file_count = 0;		
1847		file_info_ptr = lm_vector_file_ptr;		
1848		while (file_info_ptr != NULL) {		
1849		++file_count;		
1850		file_info_ptr = file_info_ptr->next;		
1851		}		
1852		write_char(lm_save_set, file_count);		
1853		file_info_ptr = lm_vector_file_ptr;		
1854		while (file_info_ptr != NULL) {		
1855		write_short(lm_save_set, file_info_ptr->instance_id);		
1856		for (i = 0; (c = file_info_ptr->name[i]) != '\0'; ++i) {		
1857		write_char(lm_save_set, c);		
1858		}		
1859		write_char(lm_save_set, c);		
1860		if (lm_get_file_size(file_info_ptr->file_ptr,		
1861		file_info_ptr->name, &filesize) == FAILURE) {		
1862		filesize = 0;		
1863		write_long(lm_save_set, filesize);		
1864		file_info_ptr = file_info_ptr->next;		
1865		}		
1866		lm_sort_def_inst_fault(lm_def_id_table, lm_inst_id_table,		
1867		lm_def_id_table_size, lm_inst_id_table_size,		
1868		&def_count, &inst_count);		
1869		/* save the definition count */		
1870		write_short(lm_save_set, def_count);		
1871		/* save the user definition id and the definition itself */		
1872		for (i = 0; i < lm_def_id_table_size; ++i) {		
1873		def = lm_def_id_table[i];		
1874		if (def == NULL)		
1875		continue;		
1876		/* save the user definition id */		
1877		write_short(lm_save_set, def->def_id);		
1878		/* save the device pattern count and the definition text */		
1879		if (lm_send_save_def_cmd(def->system_ptr,		
1880		lm_save_set, def->box_def_id) == LM_ERROR)		
1881		return (LM_ERROR);		
1882		/* save the user pid count */		
1883		user_pid_count = def->hash_table_element_count;		
1884		write_short(lm_save_set, user_pid_count);		
1885		for (i = 0; i < MAX_USER_PID_HASH_TABLE_SIZE; ++i) {		
1886		user_pid_ptr = def->user_pid_hash_table[i];		
1887		while (user_pid_ptr != NULL) {		
1888		write_long(lm_save_set, user_pid_ptr->dab_pid_table_index);		
1889		write_long(lm_save_set, user_pid_ptr->key);		
1890		user_pid_ptr = user_pid_ptr->next_in_bucket;		
1891		}		
1892		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89 TIME 1:20:53 pm	PAGE # 17/40
LINE #	SOURCE TEXT			
1918	/* First figure out how big the unspec-delay table is. */			
1919	unspec_delay_count = 0;			
1920	if (def->unspecified_delay_list != (unspecified_delay_list_element **) NULL) {			
1921	/* walk through the entire hash table. */			
1922	for (i = 0; i < MAX_UNSPECIFIED_DELAY_TABLE_SIZE; i++) {			
1923	elem_ptr = (def->unspecified_delay_list + i);			
1924	/* and count up all elements for each hash table entry. */			
1925	while (elem_ptr != (unspecified_delay_list_element **) NULL) {			
1926	unspec_delay_count++;			
1927	elem_ptr = (elem_ptr->next);			
1928	}			
1929	}			
1930	/* Now dump the table. */			
1931	write_long(lm_save_set, unspec_delay_count);			
1932	if (def->unspecified_delay_list != (unspecified_delay_list_element **) NULL) {			
1933	/* walk through the entire hash table. */			
1934	for (i = 0; i < MAX_UNSPECIFIED_DELAY_TABLE_SIZE; i++) {			
1935	elem_ptr = (def->unspecified_delay_list + i);			
1936	/* and save all elements for each hash table entry. */			
1937	while (elem_ptr != (unspecified_delay_list_element **) NULL) {			
1938	write_long(lm_save_set, elem_ptr->unspecified_delay_unspecified_value);			
1939	elem_ptr = (elem_ptr->next);			
1940	}			
1941	}			
1942	/* save inst/var count. */			
1943	write_short(lm_save_set, inst_count);			
1944	/* save the inst/fault information. */			
1945	for (i = 0; i < lm_def_id_table_size; ++i) {			
1946	def = lm_def_id_table[i];			
1947	if (def == NULL)			
1948	continue;			
1949	instance = def->instance_link;			
1950	while (instance != NULL) {			
1951	if (lm_save_instance_info(def, instance, lm_save_set) == LM_ERROR)			
1952	return (LM_ERROR);			
1953	if (lm_save_instance_pattern(instance, lm_save_set) == LM_ERROR)			
1954	return (LM_ERROR);			
1955	fault = instance->fault_link;			
1956	while (fault != NULL) {			
1957	if (lm_save_instance_info(def, fault, lm_save_set) == LM_ERROR)			
1958	return (LM_ERROR);			
1959	if (lm_save_instance_pattern(fault, lm_save_set) == LM_ERROR)			
1960	return (LM_ERROR);			
1961	fault = fault->fault_link;			
1962	}			
1963	instance = instance->link;			
1964	}			
1965	(void) lm_write_file_size(lm_save_set, filename);			
1966	(void) fclose(lm_save_set);			
1967	lm_message_types(&error_count, &warning_count);			
1968	if (error_count != 0)			
1969	return (LM_ERROR);			
1970	if (warning_count != 0)			
1971	return (LM_WARNING);			
1972	return (LM_SUCCESS);			
1973	}			
1974	lm_restore_modeler_state(filename)			
1975	char *filename;			
1976	{			
1977	long temp;			
1978	long unit;			
1979	long number;			
1980	u_long filesize;			
1981	u_short def_count;			
1982	u_short inst_count;			
1983	u_short error_count;			
1984	u_short warning_count;			
1985	u_short instance_id;			
1986	char xfilename(MAX_STRING_LENGTH);			
1987	u_char file_count;			
1988	u_char i;			
1989	u_char j;			
1990	char c;			
1991	#ifdef DEBUG			
1992	if (lm_debug[1]) {			
1993	DPRINTF(("lm_restore_modeler_state(%s)\n", filename));			
1994	}			
1995	#endif			
1996	clear_errors();			
1997	if (!lm_sim_verified) {			
1998	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1999	return (LM_ERROR);			
2000	}			
2001	if (lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)			
2002	return (LM_ERROR);			
2003	lm_save_set = NULL;			
2004	if ((lm_save_set = fopen(filename, "r")) == NULL) {			
2005	lm_queue_message(ERROR_MSG, "cannot open save-set file: %s for read",			
2006	filename);			
2007	return (LM_ERROR);			
2008	}			
2009	/* get the save-set version id */			
2010	read_long(lm_save_set, &temp);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfil.c	DATE 5/23/89	PAGE # 18/41
LINE #		SOURCE TEXT		
2039		if (temp != SAVE_SET_VERSION_ID) {		
2040		lm_queue_message(ERROR_MSG, "mismatched save-set version id, expected: %d found: %d",		
2041		SAVE_SET_VERSION_ID, temp);		
2042		(void) fclose(lm_save_set);		
2043		return (LM_ERROR);		
2044		}		
2045		/* Discard all existing data structure */		
2046		if (lm_release_definition((long) LM_ALL_DEFINITIONS) == LM_ERROR) {		
2047		(void) fclose(lm_save_set);		
2048		return (LM_ERROR);		
2049		}		
2050		/* Close all currently open Timing Measurement files */		
2051		while (lm_timing_file_ptr != NULL) {		
2052		lm_close_and_delete_file_key_id(lm_timing_file_ptr,		
2053		lm_timing_file_ptr->instance_id);		
2054		}		
2055		/* Close all currently open Test Vector files */		
2056		while (lm_vector_file_ptr != NULL) {		
2057		lm_close_and_delete_file_key_id(lm_vector_file_ptr,		
2058		lm_vector_file_ptr->instance_id);		
2059		}		
2060		}		
2061		if (lm_check_file_size(lm_save_set, filename) == LM_ERROR) {		
2062		return (LM_ERROR);		
2063		}		
2064		read_long(lm_save_set, &unit);		
2065		read_long(lm_save_set, &number);		
2066		lm_gbl_time_scale_number = -1;		
2067		}		
2068		if (lm_set_simulation_tick((unsigned long) number, unit) == LM_ERROR)		
2069		return (LM_ERROR);		
2070		}		
2071		/* Restore the simulation time */		
2072		read_long(lm_save_set, &lm_simulation_time);		
2073		}		
2074		/* Restore the logic map */		
2075		for (i = 0; i < MAX_LOGIC_STATE; ++i) {		
2076		read_short(lm_save_set, &temp);		
2077		lm_user_to_lm_logic_level_map[i].user_value = (u_short) temp;		
2078		}		
2079		read_short(lm_save_set, &temp);		
2080		lm_user_to_lm_logic_level_map[i].lm_value = (u_short) temp;		
2081		}		
2082		lm_user_to_lm_logic_level_map[i].set = 1;		
2083		}		
2084		lm_to_user_logic_level_map[temp] =		
2085		lm_user_to_lm_logic_level_map[i].user_value;		
2086		}		
2087		}		
2088		lm_logic_level_map_count = 0;		
2089		/* Restore lm_get_pin_value_type */		
2090		read_short(lm_save_set, &lm_get_pin_value_type);		
2091		}		
2092		/* Restore Timing Measurement files */		
2093		lm_timing_file_ptr = NULL;		
2094		read_char(lm_save_set, &file_count);		
2095		for (i = 0; i < file_count; ++i) {		
2096		read_short(lm_save_set, &instance_id);		
2097		j = 0;		
2098		do {		
2099		read_char(lm_save_set, &c);		
2100		xfilename[j++] = c;		
2101		} while (c != '\0');		
2102		read_long(lm_save_set, &filesize);		
2103		if (lm_append_and_insert_file(&lm_timing_file_ptr,		
2104		xfilename, instance_id, filesize) == FAILURE)		
2105		return (LM_ERROR);		
2106		}		
2107		/* Restore Test Vector file */		
2108		lm_vector_file_ptr = NULL;		
2109		read_char(lm_save_set, &file_count);		
2110		for (i = 0; i < file_count; ++i) {		
2111		read_short(lm_save_set, &instance_id);		
2112		j = 0;		
2113		do {		
2114		read_char(lm_save_set, &c);		
2115		xfilename[j++] = c;		
2116		} while (c != '\0');		
2117		read_long(lm_save_set, &filesize);		
2118		if (lm_append_and_insert_file(&lm_vector_file_ptr,		
2119		xfilename, instance_id, filesize) == FAILURE)		
2120		return (LM_ERROR);		
2121		}		
2122		/* Get the definition count */		
2123		read_short(lm_save_set, &def_count);		
2124		for (i = 0; i < def_count; ++i) {		
2125		if (lm_restore_definition(lm_save_set, filename) == LM_ERROR)		
2126		return (LM_ERROR);		
2127		}		
2128		/* Get the instance/fault count */		
2129		read_short(lm_save_set, &inst_count);		
2130		for (i = 0; i < inst_count; ++i) {		
2131		if (lm_restore_instance(lm_save_set, filename) == LM_ERROR)		
2132		return (LM_ERROR);		
2133		}		
2134		lm_definition_selected = NULL;		
2135		lm_instance_selected = NULL;		
2136		lm_last_definition_id_specified = -1;		
2137		lm_last_instance_id_specified = -1;		
2138		}		
2139		if (ferror(lm_save_set)) {		
2140		clearerr(lm_save_set);		
2141		lm_queue_message(ERROR_MSG, "error in reading from save-set file: %s during restore",		
2142		filename);		
2143		return (LM_ERROR);		
2144		}		
2145		(void) fclose(lm_save_set);		
2146		}		
2147		lm_message_types(&error_count, &warning_count);		
2148		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 19/42
LINE #		SOURCE TEXT		
2159		if (error_count != 0)		
2160		return (LM_ERROR);		
2161		}		
2162		if (warning_count != 0)		
2163		return (LM_WARNING);		
2164		}		
2165		return (LM_SUCCESS);		
2166		}		
2167		lm_get_message(attributes, string)		
2168		long *attribute;		
2169		char **string;		
2170		{		
2171		u_short type;		
2172		if (lm_dequeue_message(&type, lm_get_error_string) != FAILURE) {		
2173		{		
2174		*string = lm_get_error_string;		
2175		if (type == WARNING_MSG)		
2176		*attribute = LM_WARNING;		
2177		else		
2178		*attribute = LM_ERROR;		
2179		} else {		
2180		*string = NULL;		
2181		*attribute = LM_NO_MORE_MESSAGES;		
2182		}		
2183		}		
2184		#ifdef DEBUG		
2185		if (lm_debug(2)) {		
2186		DPRINTF(("lm_get_message(%d, %s)\n", *attribute, *string));		
2187		}		
2188		#endif		
2189		return (LM_SUCCESS);		
2190		}		
2191		}		
2192		lm_timing_measurement(instance_or_fault_id, command, filename)		
2193		long instance_or_fault_id;		
2194		long command;		
2195		char *filename;		
2196		{		
2197		FILE_INFO *file_ptr;		
2198		u_short error_count;		
2199		u_short warning_count;		
2200		char wait_string[256];		
2201		}		
2202		#ifdef DEBUG		
2203		if (lm_debug(1)) {		
2204		DPRINTF(("lm_timing_measurement(%d, %d, %s)\n", instance_or_fault_id,		
2205		command, filename));		
2206		}		
2207		#endif		
2208		clear_errors();		
2209		if (!lm_sim_verified) {		
2210		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
2211		return (LM_ERROR);		
2212		}		
2213		if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
2214		return (LM_ERROR);		
2215		}		
2216		switch (command) {		
2217		case LM_TIMING_OFF:		
2218		lm_instance_selected->tm_file = NULL;		
2219		if (lm_send_times_cmd(lm_instance_selected,		
2220		(u_char) command) == FAILURE)		
2221		return (LM_ERROR);		
2222		lm_message_types(&error_count, &warning_count);		
2223		if (error_count != 0)		
2224		return (LM_ERROR);		
2225		if (warning_count != 0)		
2226		return (LM_WARNING);		
2227		return (LM_SUCCESS);		
2228		case LM_TIMING_ON:		
2229		break;		
2230		default:		
2231		lm_queue_message(ERROR_MSG, "internal simulator error: illegal command: %d specified",		
2232		command);		
2233		return (LM_ERROR);		
2234		}		
2235		if (filename == LM_NO_FILE) {		
2236		if (lm_search_key_id(lm_timing_file_ptr,		
2237		(u_short) instance_or_fault_id, &file_ptr) == SUCCESS) {		
2238		lm_close_and_delete_file_key_id(&lm_timing_file_ptr,		
2239		(u_short) instance_or_fault_id);		
2240		lm_instance_selected->tm_file = NULL;		
2241		}		
2242		} else {		
2243		if (lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)		
2244		return (LM_ERROR);		
2245		if (lm_check_file_type(filename) == FAILURE) {		
2246		return (LM_ERROR);		
2247		}		
2248		if (lm_search_key_id(lm_timing_file_ptr,		
2249		(u_short) instance_or_fault_id, &file_ptr) == SUCCESS) {		
2250		if (strcmp(file_ptr->name, filename) != 0) {		
2251		lm_close_and_delete_file_key_id(&lm_timing_file_ptr,		
2252		(u_short) instance_or_fault_id);		
2253		lm_instance_selected->tm_file = NULL;		
2254		} else {		
2255		if (lm_send_times_cmd(lm_instance_selected,		
2256		(u_char) command) == FAILURE)		
2257		return (LM_ERROR);		
2258		lm_instance_selected->tm_file = file_ptr->file_ptr;		
2259		lm_message_types(&error_count, &warning_count);		
2260		if (error_count != 0)		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 20/43
LINE #		SOURCE TEXT		
2279		return (LM_ERROR);		
2280		if (warning_count != 0)		
2281		return (LM_WARN);		
2282		return (LM_SUCCESS);		
2283		}		
2284		}		
2285		/*		
2286		Make sure that the filename is not used by other instance, or used for		
2287		vector file.		
2288		*/		
2289		if (lm_search_key_name(lm_timing_file_ptr, filename) == SUCCESS) {		
2290		lm_queue_message(ERROR_MSG, "Filename: %s is used by other instance",		
2291		filename);		
2292		return (LM_ERROR);		
2293		}		
2294		if (lm_search_key_name(lm_vector_file_ptr, filename) == SUCCESS) {		
2295		lm_queue_message(ERROR_MSG, "Filename: %s is used for vector logging",		
2296		filename);		
2297		return (LM_ERROR);		
2298		}		
2299		if (lm_open_and_insert_file(&lm_timing_file_ptr, filename,		
2300		(u_short) instance_or_fault_id) == FAILURE)		
2301		return (LM_ERROR);		
2302		else {		
2303		lm_instance_selected->tm_file = lm_timing_file_ptr->file_ptr;		
2304		switch (lm_get_time_scale_units) {		
2305		case LM_FEMTOSECONDS:		
2306		(void) sprintf(unit_string, "femtoseconds");		
2307		break;		
2308		case LM_PICOSECONDS:		
2309		(void) sprintf(unit_string, "picoseconds");		
2310		break;		
2311		case LM_NANOSECONDS:		
2312		(void) sprintf(unit_string, "nanoseconds");		
2313		break;		
2314		case LM_MICROSECONDS:		
2315		(void) sprintf(unit_string, "microseconds");		
2316		break;		
2317		default:		
2318		(void) sprintf(unit_string, "microseconds");		
2319		break;		
2320		}		
2321		(void) fprintf(lm_instance_selected->tm_file,		
2322		"device: %s, instance: %d, simtime: %s, info: %s\n",		
2323		lm_def_id_table[lm_instance_selected->def_id]->device_name,		
2324		lm_instance_selected->inst_id,		
2325		unit_string,		
2326		lm_instance_selected->device_info_string);		
2327		}		
2328		}		
2329		if (lm_send_time_cmd(lm_instance_selected, (u_char) command) == FAILURE)		
2330		return (LM_ERROR);		
2331		lm_message_type(&error_count, &warning_count);		
2332		if (error_count != 0)		
2333		return (LM_ERROR);		
2334		if (warning_count != 0)		
2335		return (LM_WARNING);		
2336		return (LM_SUCCESS);		
2337		}		
2338		lm_loop_patterns(instance_or_fault_id, command)		
2339		loop		
2340		loop		
2341		{		
2342		char		
2343		inst		
2344		state,		
2345		status;		
2346		{		
2347		#ifdef DEBUG		
2348		if (lm_debug[1]) {		
2349		fprintf(stderr, "lm_loop_patterns(%d, %d)\n", instance_or_fault_id, command);		
2350		}		
2351		#endif		
2352		clear_errors();		
2353		if (!lm_sim_verified) {		
2354		lm_queue_message(ERROR_MSG,		
2355		"internal simulator error: lm_begin_modeling_session() has not yet been called");		
2356		return (LM_ERROR);		
2357		}		
2358		if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
2359		return (LM_ERROR);		
2360		}		
2361		switch (command) {		
2362		case LM_LOOP_ON:		
2363		/* start loop mode */		
2364		lm_choose_conn(lm_instance_selected->box_id);		
2365		lm_put_inst(LOOP_PTN_CMD);		
2366		lm_put_short(lm_instance_selected->box_inst_id);		
2367		if (!lm_end_put(lm_instance_selected->box_id)) {		
2368		return (LM_ERROR);		
2369		}		
2370		if (lm_get_inst() != LOOP_PTN_ANS) {		
2371		lm_queue_message(ERROR_MSG,		
2372		"internal error: Wrong reply received from modeler: %s",		
2373		lm_def_id_table[lm_instance_selected->def_id]->system_ptr->name);		
2374		return (LM_ERROR);		
2375		}		
2376		return lm_dequeue_errors();		
2377		}		
2378		case LM_LOOP_OFF:		
2379		status = lm_request_interrupt(lm_def_id_table[lm_instance_selected->def_id]		
2380		->system_ptr->name, &status);		
2381		if ((status == LM_WARNING) && (status == RUNNING_LOOPMODE)) {		
2382		lm_remove_message(WARNING_MSG, "modeler running loop mode");		
2383		status = LM_SUCCESS;		
2384		}		
2385		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 21/44
LINE #		SOURCE TEXT		
2399		} else if (status == LM_SUCCESS) {		
2400		lm_queue_message(WARNING_MSG, "modeler not running loop mode");		
2401		status = LM_WARNING;		
2402		return (status);		
2403		}		
2404		default:		
2405		lm_queue_message(ERROR_MSG, "internal simulator error: illegal command: td specified",		
2406		command);		
2407		return (LM_ERROR);		
2408		}		
2409		}		
2410		}		
2411		lm_specify_instance_id(instance_or_fault_id)		
2412		register long instance_or_fault_id,		
2413		{		
2414		register INSTANCE_INFO *instance;		
2415		{		
2416		if ((instance_or_fault_id != lm_last_instance_id_specified)		
2417		(instance_or_fault_id == -1)) {		
2418		{		
2419		if ((instance_or_fault_id < 0)		
2420		(instance_or_fault_id >= lm_inst_id_table_size)) {		
2421		lm_queue_message(ERROR_MSG,		
2422		"internal simulator error: invalid instance/fault id: td specified",		
2423		instance_or_fault_id);		
2424		return (LM_ERROR);		
2425		}		
2426		instance = lm_inst_id_table(instance_or_fault_id);		
2427		if (BOGUS_INSTANCE(instance)) {		
2428		lm_queue_message(ERROR_MSG,		
2429		"internal simulator error: invalid instance/fault id: td specified",		
2430		instance_or_fault_id);		
2431		return (LM_ERROR);		
2432		}		
2433		lm_last_instance_id_specified = instance_or_fault_id;		
2434		lm_instance_selected = instance;		
2435		}		
2436		return (SUCCESS);		
2437		}		
2438		}		
2439		lm_reset_instance(instance_id)		
2440		long instance_id,		
2441		{		
2442		INSTANCE_INFO *instance;		
2443		INSTANCE_INFO *temp;		
2444		FILE_INFO *file_ptr;		
2445		PIN_INFO *pin;		
2446		long ret;		
2447		u_short i;		
2448		u_short out_count;		
2449		short pin_no;		
2450		{		
2451		#ifdef DEBUG		
2452		if (lm_debug(1)) {		
2453		PRINTF("lm_reset_instance(td)\n", instance_id);		
2454		}		
2455		#endif		
2456		{		
2457		clear_errors();		
2458		{		
2459		if (!lm_sim_verified) {		
2460		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
2461		return (LM_ERROR);		
2462		}		
2463		lm_last_instance_id_specified = -1;		
2464		lm_instance_selected = NULL;		
2465		{		
2466		if (instance_id == LM_ALL_INSTANCES) {		
2467		{		
2468		lm_simulation_time = 0;		
2469		{		
2470		/*		
2471		* Go through the instance table and reset all INSTANCES. The faults for		
2472		* All instances will be released in the else part of the if statement		
2473		* above.		
2474		*/		
2475		for (i = 0; i < lm_inst_id_table_size; ++i) {		
2476		instance = lm_inst_id_table[i];		
2477		{		
2478		if (BOGUS_INSTANCE(instance))		
2479		continue;		
2480		{		
2481		if (instance->assoc_box_inst_id == -1) {		
2482		/* If it is an INSTANCE then reset it. */		
2483		if ((ret = lm_reset_instance((long) i)) == LM_ERROR)		
2484		return (LM_ERROR);		
2485		}		
2486		}		
2487		return (ret);		
2488		}		
2489		else {		
2490		if ((instance_id < 0)		
2491		(instance_id >= lm_inst_id_table_size)) {		
2492		lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: td specified",		
2493		instance_id);		
2494		return (LM_ERROR);		
2495		}		
2496		instance = lm_inst_id_table(instance_id);		
2497		if (BOGUS_INSTANCE(instance)) {		
2498		lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: td specified",		
2499		instance_id);		
2500		return (LM_ERROR);		
2501		}		
2502		if (instance->assoc_box_inst_id != -1) {		
2503		lm_queue_message(ERROR_MSG, "internal simulator error: cannot reset a fault, fault id: td",		
2504		instance_id);		
2505		return (LM_ERROR);		
2506		}		
2507		/* Release all faults associated with this instance */		
2508		for (i = 0; i < lm_inst_id_table_size; ++i) {		
2509		temp = lm_inst_id_table[i];		
2510		{		
2511		if (BOGUS_INSTANCE(temp))		
2512		continue;		
2513		{		
2514		if (temp->assoc_box_inst_id == instance->box_inst_id) {		
2515		if (lm_release_fault((long) i) == LM_ERROR)		
2516		return (LM_ERROR);		
2517		}		
2518		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 22/45
LINE #		SOURCE TEXT		
2519		if (lm_search_key_id(lm_timing_file_ptr,		
2520		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
2521		close_and_delete_file_key_id(lm_timing_file_ptr,		
2522		(u_short) instance->inst_id,		
2523		instance->tm_file = NULL;		
2524		}		
2525		if (lm_search_key_id(lm_vector_file_ptr,		
2526		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
2527		lm_output_last_vector(instance),		
2528		lm_close_and_delete_file_key_id(lm_vector_file_ptr,		
2529		(u_short) instance->inst_id,		
2530		instance->vector_file = NULL;		
2531		}		
2532		if (lm_reinitialize_instance(instance) == FAILURE) {		
2533		return (LM_ERROR);		
2534		}		
2535		lm_choose_conn(instance->box_id);		
2536		lm_put_inst(RESET_INST_CMD);		
2537		lm_put_short(instance->box_inst_id);		
2538		if (lm_end_put(instance->box_id))		
2539		return (LM_ERROR);		
2540		}		
2541		if (lm_get_inst() != RESET_INST_ANS) {		
2542		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
2543		lm_def_id_table(instance->def_id->system_ptr->name);		
2544		return (LM_ERROR);		
2545		}		
2546		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
2547		return (LM_ERROR);		
2548		}		
2549		out_count = lm_get_short();		
2550		for (i = 0; i < out_count; ++i) {		
2551		pin_no = lm_get_short();		
2552		pin = instance->pin_table[pin_no];		
2553		pin->out_value = lm_get_char();		
2554		}		
2555		}		
2556		lm_simulation_time = 0;		
2557		return (ret);		
2558		}		
2559		/*		
2560		lm_log_test_vectors(instance_or_fault_id, command, filename)		
2561		long instance_or_fault_id,		
2562		long command,		
2563		char *filename,		
2564		{		
2565		FILE INFO *file_ptr;		
2566		u_long save_simulation_time;		
2567		}		
2568		#ifdef DEBUG		
2569		if (lm_debug[1]) {		
2570		fprintf(stderr, "lm_log_test_vectors(%d, %d, %s)\n",		
2571		instance_or_fault_id, command, filename);		
2572		}		
2573		#endif		
2574		clear_errors();		
2575		if (!lm_sim_verified) {		
2576		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
2577		return (LM_ERROR);		
2578		}		
2579		if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
2580		return (LM_ERROR);		
2581		}		
2582		switch (command) {		
2583		case LM_LOGGING_OFF:		
2584		if (lm_instance_selected->vector_file != NULL) {		
2585		lm_output_last_vector(lm_instance_selected);		
2586		lm_close_and_delete_file_key_id(lm_vector_file_ptr,		
2587		(u_short) instance_or_fault_id,		
2588		lm_instance_selected->vector_file = NULL;		
2589		}		
2590		return (LM_SUCCESS);		
2591		case LM_LOGGING_ON:		
2592		break;		
2593		default:		
2594		lm_queue_message(ERROR_MSG, "internal simulator error: illegal command: %d specified",		
2595		command);		
2596		return (LM_ERROR);		
2597		}		
2598		if (lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)		
2599		return (LM_ERROR);		
2600		if (lm_check_file_type(filename) == FAILURE) {		
2601		return (LM_ERROR);		
2602		}		
2603		if (lm_search_key_id(lm_vector_file_ptr,		
2604		(u_short) instance_or_fault_id, &file_ptr) == SUCCESS) {		
2605		if (strcmp(file_ptr->name, filename) != 0) {		
2606		lm_instance_selected->vector_file = NULL;		
2607		lm_close_and_delete_file_key_id(lm_vector_file_ptr,		
2608		(u_short) instance_or_fault_id,		
2609		}		
2610		else {		
2611		lm_instance_selected->vector_file = file_ptr->file_ptr;		
2612		return (LM_SUCCESS);		
2613		}		
2614		}		
2615		/*		
2616		* Make sure that the filename is NOT used by other instance, or used for		
2617		* timing measurement file.		
2618		*/		
2619		if (lm_search_key_name(lm_timing_file_ptr, filename) == SUCCESS) {		
2620		lm_queue_message(ERROR_MSG, "filename: %s is used for timing measurement capture",		
2621		filename);		
2622		return (LM_ERROR);		
2623		}		
2624		if (lm_search_key_name(lm_vector_file_ptr, filename) == SUCCESS) {		
2625		lm_queue_message(ERROR_MSG, "filename: %s is used by another instance",		
2626		filename);		
2627		return (LM_ERROR);		
2628		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi1.c	DATE 5/23/89	PAGE # 23/46
LINE #		SOURCE TEXT		
2639		if (lm_open_and_insert_file(&lm_vector_file_ptr, filename,		
2640		(u_short) instance_or_fault_id) == FAILURE)		
2641		return (LM_ERROR);		
2642		else {		
2643		lm_instance_selected->vector_file = lm_vector_file_ptr->file_ptr;		
2644		}		
2645		if (lm_write_vector_file_header(lm_instance_selected) == FAILURE) {		
2646		lm_close_and_delete_file_key_id(&lm_vector_file_ptr,		
2647		(u_short) instance_or_fault_id);		
2648		lm_instance_selected->vector_file = NULL;		
2649		return (LM_ERROR);		
2650		}		
2651		lm_update_vector_buffer(lm_instance_selected);		
2652		return (LM_SUCCESS);		
2653				
2654				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 1/47
LINE #	SOURCE TEXT			
1	/* SCS ID: sfi2.c rev 3.2, 5/2/89 at 17:23:23 */			
2	#include <stdio.h>			
3	#include <ctype.h>			
4	#include "common.h"			
5	#include "network.h"			
6	#include "message.h"			
7	#include "lm_sfi.h"			
8	#include "lsfi.h"			
9	#include "protans.h"			
10	#include "protcmd.h"			
11	#include "conv.h"			
12	#include "globals.h"			
13	extern char *lm_malloc();			
14	extern char *lm_realloc();			
15	extern char *lm_malloc();			
16	extern void lm_init_vars();			
17	/* MODEL entry points */			
18	extern char *lm_begin_modeling_session();			
19	extern void lm_init_vars();			
20	lm_pattern_history(instance_id, command)			
21	long instance_id,			
22	long command;			
23	{			
24	INSTANCE_INFO *instance;			
25	long ret;			
26	u_short i;			
27	{			
28	#ifdef DEBUG			
29	if (lm_debug[1]) {			
30	DPRINTF(("lm_pattern_history(%d, %d)\n", instance_id, command));			
31	}			
32	#endif			
33	clear_errors();			
34	if (!lm_sim_verified) {			
35	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
36	return (LM_ERROR);			
37	}			
38	if (instance_id == LM_ALL_INSTANCES) {			
39	for (i = 0; i < lm_inst_id_table_size; ++i) {			
40	instance = lm_inst_id_table[i];			
41	if (BOGUS_INSTANCE(instance))			
42	continue;			
43	if (lm_pattern_history((long) i, command) == LM_ERROR)			
44	return (LM_ERROR);			
45	}			
46	} else {			
47	if ((instance_id < 0)			
48	(instance_id >= lm_inst_id_table_size)) {			
49	lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: %d specified",			
50	instance_id);			
51	return (LM_ERROR);			
52	}			
53	instance = lm_inst_id_table[instance_id];			
54	if (BOGUS_INSTANCE(instance)) {			
55	lm_queue_message(ERROR_MSG, "internal simulator error: invalid instance id: %d specified",			
56	instance_id);			
57	return (LM_ERROR);			
58	}			
59	lm_choose_conn(instance->box_id);			
60	lm_put_int(PTM_HIST_CMD);			
61	lm_put_short(instance->box_inst_id);			
62	lm_put_char(command);			
63	if (!lm_end_put(instance->box_id))			
64	return (LM_ERROR);			
65	if (lm_get_int() != PTM_HIST_ANS) {			
66	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",			
67	lm_def_id_table(instance->def_id)->system_ptr->name);			
68	return (LM_ERROR);			
69	}			
70	if ((ret = lm_dequeue_errors()) == LM_ERROR)			
71	return (LM_ERROR);			
72	}			
73	return (ret);			
74	}			
75	/*			
76	lm_inquire_modeler_list(number_of_modelers, list_of_modelers)			
77	long number_of_modelers;			
78	char **list_of_modelers[];			
79	{			
80	u_short error_count;			
81	u_short warning_count;			
82	u_short index;			
83	u_short i;			
84	{			
85	#ifdef DEBUG			
86	if (lm_debug[1]) {			
87	DPRINTF(("lm_inquire_modeler_list(%d, %d)\n",			
88	number_of_modelers, list_of_modelers));			
89	}			
90	#endif			
91	clear_errors();			
92	*list_of_modelers = NULL;			
93	if (!lm_sim_verified) {			
94	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
95	return (LM_ERROR);			
96	}			
97	for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i)			
98	if (lm_modeler_list_buf[i] != NULL) {			
99	FREE(lm_modeler_list_buf[i]);			
100	lm_modeler_list_buf[i] = NULL;			
101	}			
102	*number_of_modelers = 0;			
103	for (index = 0; i = 0; lm_system_table[i] != NULL; ++i) {			
104	{			
105	if (lm_modeler_list_buf[i] != NULL) {			
106	FREE(lm_modeler_list_buf[i]);			
107	lm_modeler_list_buf[i] = NULL;			
108	}			
109	}			
110	}			
111	}			
112	}			
113	}			
114	}			
115	}			
116	}			
117	}			
118	}			
119	}			
120	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 2/48
LINE #	SOURCE TEXT			
121	if (strlen(lm_system_table[i]->name) == 0)			
122	continue;			
123	if (index == MAX_SYSTEM_TABLE_SIZE) {			
124	lm_queue_message(ERROR_MSG, "internal error: return space is full");			
125	return (LM_ERROR);			
126	}			
127	lm_modeler_list_buf[index] =			
128	DUALLOC((unsigned)(strlen(lm_system_table[i]->name) + 1));			
129	if (lm_modeler_list_buf[index] == NULL) {			
130	lm_queue_message(ERROR_MSG, "out of memory on host");			
131	return (LM_ERROR);			
132	}			
133	(void) strcpy(lm_modeler_list_buf[index], lm_system_table[i]->name);			
134	++index;			
135	++("number_of_modelers");			
136	}			
137	*list_of_modelers = lm_modeler_list_buf;			
138	#ifdef DEBUG			
139	if (lm_debug[2]) {			
140	DPRINTF(("lm_inquire_modeler_list(%d, ", "number_of_modelers));			
141	if ("number_of_modelers" {			
142	for (i = 0; i < "number_of_modelers - 1; ++i) {			
143	DPRINTF(("ts, ", lm_modeler_list_buf[i]);			
144	}			
145	DPRINTF(("ts\n", lm_modeler_list_buf[i]);			
146	}			
147	#endif			
148	lm_message_types(terror_count, twarning_count);			
149	if (error_count != 0)			
150	return (LM_ERROR);			
151	if (warning_count != 0)			
152	return (LM_WARNING);			
153	return (LM_SUCCESS);			
154	}			
155	lm_inquire_modeler(attribute, value)			
156	long attribute;			
157	long *value;			
158	{			
159	long ret;			
160	#ifdef DEBUG			
161	if (lm_debug[1]) {			
162	DPRINTF(("lm_inquire_modeler(%d, %dx)\n", attribute, value));			
163	}			
164	#endif			
165	clear_errors();			
166	if (!lm_sim_verified) {			
167	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
168	return (LM_ERROR);			
169	}			
170	*value = 0;			
171	if (lm_system_selected == NULL) {			
172	lm_queue_message(ERROR_MSG, "no modeler has been selected");			
173	return (LM_ERROR);			
174	}			
175	lm_choose_conn(lm_system_selected->sg);			
176	lm_put_int(IMP_MODELER_CMD);			
177	lm_put_int(attribute);			
178	if (!lm_end_put(lm_system_selected->sg))			
179	return (LM_ERROR);			
180	if (lm_get_int() != IMP_MODELER_MSG) {			
181	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: ts",			
182	lm_system_selected->name);			
183	return (LM_ERROR);			
184	}			
185	if ((ret = lm_dequeue_errors()) == LM_ERROR)			
186	return (LM_ERROR);			
187	*value = lm_get_int();			
188	#ifdef DEBUG			
189	if (lm_debug[2]) {			
190	DPRINTF(("lm_inquire_modeler(%d, %d)\n", attribute, *value));			
191	}			
192	#endif			
193	return (ret);			
194	}			
195	lm_password(attribute, password)			
196	long attribute;			
197	char *password;			
198	{			
199	long ret;			
200	char c;			
201	#ifdef DEBUG			
202	if (lm_debug[1]) {			
203	DPRINTF(("lm_password(%d, %s)\n", attribute, password));			
204	}			
205	#endif			
206	clear_errors();			
207	if (!lm_sim_verified) {			
208	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
209	return (LM_ERROR);			
210	}			
211	if (lm_system_selected == NULL) {			
212	lm_queue_message(ERROR_MSG, "no modeler has been selected");			
213	return (LM_ERROR);			
214	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 3/49
LINE #		SOURCE TEXT		
241		lm_choose_conn(lm_system_selected->fd);		
242		lm_put_int(PASSWORD_CMD);		
243		lm_put_int(attribute);		
244		while (c = *password++) {		
245		lm_put_char(c);		
246		}		
247		lm_put_char("\0");		
248		if (!lm_end_put(lm_system_selected->fd))		
249		return (LM_ERROR);		
250		if (lm_get_int() != PASSWORD_ANS) {		
251		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
252		lm_system_selected->name);		
253		return (LM_ERROR);		
254		}		
255		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
256		return (LM_ERROR);		
257		#ifdef DEBUG		
258		if (lm_debug[2]) {		
259		DPRINTF(("lm_password(%d, %d)\n"),		
260		}		
261		#endif		
262		return (ret);		
263		}		
264		lm_inquire_user_list(number_of_users, list_of_users,		
265		list_of_hostnames, list_of_usernames)		
266		long *number_of_users;		
267		long *list_of_users[];		
268		char **list_of_hostnames[];		
269		char **list_of_usernames[];		
270		{		
271		char *ptr;		
272		long ret;		
273		u_short i;		
274		u_short j;		
275		char c;		
276		#ifdef DEBUG		
277		if (lm_debug[1]) {		
278		DPRINTF(("lm_inquire_user_list(%d, %d, %d, %d)\n",		
279		number_of_users, list_of_users,		
280		list_of_hostnames, list_of_usernames));		
281		}		
282		#endif		
283		clear_errors();		
284		if (!lm_sim_verified) {		
285		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
286		return (LM_ERROR);		
287		}		
288		*number_of_users = 0;		
289		if (lm_system_selected == NULL) {		
290		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
291		return (LM_ERROR);		
292		}		
293		lm_choose_conn(lm_system_selected->fd);		
294		lm_put_int(IMQ_USER_LIST_CMD);		
295		if (!lm_end_put(lm_system_selected->fd))		
296		return (LM_ERROR);		
297		if (lm_get_int() != IMQ_USER_LIST_ANS) {		
298		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
299		lm_system_selected->name);		
300		return (LM_ERROR);		
301		}		
302		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
303		return (LM_ERROR);		
304		*number_of_users = lm_get_int();		
305		for (i = 0; i < *number_of_users; ++i)		
306		lm_iul_users[i] = lm_get_int();		
307		for (i = 0; i < *number_of_users; ++i) {		
308		ptr = lm_iul_hostnames[i];		
309		for (j = 0; (c = lm_get_char()) != '\0'; ++j)		
310		ptr[j] = c;		
311		ptr[j] = '\0';		
312		}		
313		for (i = 0; i < *number_of_users; ++i) {		
314		ptr = lm_iul_usernames[i];		
315		for (j = 0; (c = lm_get_char()) != '\0'; ++j)		
316		ptr[j] = c;		
317		ptr[j] = '\0';		
318		}		
319		*list_of_users = lm_iul_users;		
320		*list_of_hostnames = lm_iul_hostnames;		
321		*list_of_usernames = lm_iul_usernames;		
322		#ifdef DEBUG		
323		if (lm_debug[2]) {		
324		if (*number_of_users) {		
325		DPRINTF(("lm_inquire_user_list(%d, ", *number_of_users);		
326		for (i = 0; i < *number_of_users - 1; ++i) {		
327		DPRINTF((" %d %s ", lm_iul_users[i],		
328		lm_iul_hostnames[i], lm_iul_usernames[i]);		
329		}		
330		DPRINTF((" %d %s)\n", lm_iul_users[i],		
331		lm_iul_hostnames[i], lm_iul_usernames[i]);		
332		}		
333		else {		
334		DPRINTF(("lm_inquire_user_list(%d)\n", *number_of_users);		
335		}		
336		#endif		
337		return (ret);		
338		}		
339		}		
340		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 4/50
LINE #		SOURCE TEXT		
351		lm_inquire_user(user_number, attribute, value)		
352		long user_number,		
353		long attribute,		
354		long value,		
355		{		
356		long ret,		
357		{		
358		#ifdef DEBUG		
359		if (lm_debug[1]) {		
360		DPRINTF(("lm_inquire_user(%d, %d, %d)\n",		
361		user_number, attribute, value));		
362		}		
363		#endif		
364		clear_errors();		
365		if (!lm_sim_verified) {		
366		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
367		return (LM_ERROR);		
368		}		
369		value = 0;		
370		if (lm_system_selected == NULL) {		
371		lm_queue_message(ERROR_MSG, "no modular has been selected");		
372		return (LM_ERROR);		
373		}		
374		lm_choose_conn(lm_system_selected->id);		
375		lm_put_int(IMG_USER_CMD);		
376		lm_put_int(user_number);		
377		lm_put_int(attribute);		
378		if (!lm_end_put(lm_system_selected->id))		
379		return (LM_ERROR);		
380		if (lm_get_int() != IMG_USER_ANS) {		
381		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modular: %s",		
382		lm_system_selected->name);		
383		return (LM_ERROR);		
384		}		
385		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
386		return (LM_ERROR);		
387		value = lm_get_int();		
388		#ifdef DEBUG		
389		if (lm_debug[2]) {		
390		DPRINTF(("lm_inquire_user(%d, %d, %d)\n",		
391		value));		
392		}		
393		#endif		
394		return (ret);		
395		}		
396		lm_inquire_lane(lane_number, attribute, value)		
397		long lane_number,		
398		long attribute,		
399		long value,		
400		{		
401		long ret,		
402		{		
403		#ifdef DEBUG		
404		if (lm_debug[1]) {		
405		DPRINTF(("lm_inquire_lane(%d, %d, %d)\n",		
406		lane_number, attribute, value));		
407		}		
408		#endif		
409		clear_errors();		
410		if (!lm_sim_verified) {		
411		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
412		return (LM_ERROR);		
413		}		
414		value = 0;		
415		if (lm_system_selected == NULL) {		
416		lm_queue_message(ERROR_MSG, "no modular has been selected");		
417		return (LM_ERROR);		
418		}		
419		lm_choose_conn(lm_system_selected->id);		
420		lm_put_int(IMG_LANE_CMD);		
421		lm_put_int(lane_number);		
422		lm_put_int(attribute);		
423		if (!lm_end_put(lm_system_selected->id))		
424		return (LM_ERROR);		
425		if (lm_get_int() != IMG_LANE_ANS) {		
426		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modular: %s",		
427		lm_system_selected->name);		
428		return (LM_ERROR);		
429		}		
430		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
431		return (LM_ERROR);		
432		value = lm_get_int();		
433		#ifdef DEBUG		
434		if (lm_debug[2]) {		
435		DPRINTF(("lm_inquire_lane(%d, %d, %d)\n",		
436		value));		
437		}		
438		#endif		
439		return (ret);		
440		}		
441		lm_inquire_pam(lane_number, pam_number, attribute, value)		
442		long lane_number,		
443		long pam_number,		
444		long attribute,		
445		long value,		
446		{		
447		long ret,		
448		{		
449		#ifdef DEBUG		
450		if (lm_debug[1]) {		
451		DPRINTF(("lm_inquire_pam(%d, %d, %d, %d)\n",		
452		lane_number, pam_number, attribute, value));		
453		}		
454		#endif		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 5/51
LINE #		SOURCE TEXT		
481		}		
482		#endif		
483		clear_errors();		
484				
485		if (!lm_sim_verified) {		
486		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
487		return (LM_ERROR);		
488		}		
489		*value = 0;		
490				
491		if (lm_system_selected == NULL) {		
492		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
493		return (LM_ERROR);		
494		}		
495		lm_choose_conn(lm_system_selected->id);		
496		lm_put_int(IMG_PAN_CMD);		
497		lm_put_int(lame_number);		
498		lm_put_int(pam_number);		
499		lm_put_int(attribute);		
500				
501		if (!lm_end_put(lm_system_selected->id))		
502		return (LM_ERROR);		
503		if (lm_get_int() != IMG_PAN_ANS) {		
504		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
505		lm_system_selected->name);		
506		return (LM_ERROR);		
507		}		
508		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
509		return (LM_ERROR);		
510				
511		*value = lm_get_int();		
512				
513		#ifdef DEBUG		
514		if (lm_debug[2]) {		
515		DPRINTF(("lm_inquire_pam(%d, %d, %d, %d)\n",		
516		*value));		
517		}		
518		#endif		
519		return (ret);		
520		}		
521				
522		lm_inquire_pel(lame, slot, attribute, value)		
523		long lame;		
524		long slot;		
525		long attribute;		
526		long *value;		
527		{		
528		long ret;		
529		}		
530		#ifdef DEBUG		
531		if (lm_debug[1]) {		
532		DPRINTF(("lm_inquire_pel(%d, %d, %d, %d)\n",		
533		lame, slot, attribute, value));		
534		}		
535		#endif		
536		clear_errors();		
537				
538		if (!lm_sim_verified) {		
539		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
540		return (LM_ERROR);		
541		}		
542		*value = 0;		
543				
544		if (lm_system_selected == NULL) {		
545		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
546		return (LM_ERROR);		
547		}		
548		lm_choose_conn(lm_system_selected->id);		
549		lm_put_int(IMG_PEL_CMD);		
550		lm_put_int(lame);		
551		lm_put_int(slot);		
552		lm_put_int(attribute);		
553				
554		if (!lm_end_put(lm_system_selected->id))		
555		return (LM_ERROR);		
556		if (lm_get_int() != IMG_PEL_ANS) {		
557		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
558		lm_system_selected->name);		
559		return (LM_ERROR);		
560		}		
561		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
562		return (LM_ERROR);		
563				
564		*value = lm_get_int();		
565				
566		#ifdef DEBUG		
567		if (lm_debug[2]) {		
568		DPRINTF(("lm_inquire_pel(%d, %d, %d, %d)\n",		
569		*value));		
570		}		
571		#endif		
572		return (ret);		
573		}		
574				
575		lm_inquire_device_list(number_of_devices, list_of_device_numbers,		
576		list_of_device_names)		
577		long number_of_devices;		
578		long list_of_device_numbers[];		
579		char list_of_device_names[];		
580		{		
581		char *temp_ptr;		
582		long ret;		
583		u_short i;		
584		u_short j;		
585		char c;		
586		#ifdef DEBUG		
587		if (lm_debug[1]) {		
588		DPRINTF(("lm_inquire_device_list(%d, %d, %d)\n",		
589		number_of_devices, list_of_device_numbers,		
590		list_of_device_names));		
591		}		
592		#endif		
593		}		
594		#endif		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 6/52
LINE #		SOURCE TEXT		
601		clear_errors();		
602		if (!lm_sim_verified) {		
603		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
604		return (LM_ERROR);		
605		}		
606		number_of_devices = 0;		
607		if (lm_system_selected == NULL) {		
608		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
609		return (LM_ERROR);		
610		}		
611		lm_choose_coas(lm_system_selected->id);		
612		lm_put_int(IMQ_DEVICE_LIST_CMD);		
613		if (!lm_end_put(lm_system_selected->id))		
614		return (LM_ERROR);		
615		if (lm_get_int() != IMQ_DEVICE_LIST_ANS) {		
616		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
617		lm_system_selected->name);		
618		return (LM_ERROR);		
619		}		
620		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
621		return (LM_ERROR);		
622		}		
623		number_of_devices = lm_get_int();		
624		/* Get the device numbers from the Network buffer */		
625		for (i = 0; i < number_of_devices; ++i) {		
626		lm_idl_device_numbers[i] = lm_get_int();		
627		}		
628		/* Get the device names from the Network buffer */		
629		for (i = 0; i < number_of_devices; ++i) {		
630		temp_ptr = lm_idl_device_names[i];		
631		for (j = 0; (c = lm_get_char()) != '\0'; ++j)		
632		temp_ptr[j] = c;		
633		temp_ptr[j] = '\0';		
634		}		
635		list_of_device_numbers = lm_idl_device_numbers;		
636		list_of_device_names = lm_idl_device_names;		
637		#ifdef DEBUG		
638		if (lm_debug[2]) {		
639		if (number_of_devices) {		
640		PRINTF("lm_inquire_device_list(%d, ", number_of_devices);		
641		for (i = 0; i < number_of_devices - 1; ++i) {		
642		PRINTF("%d ",		
643		lm_idl_device_numbers[i], lm_idl_device_names[i]);		
644		}		
645		PRINTF("%d %s", lm_idl_device_numbers[i], lm_idl_device_names[i]);		
646		}		
647		else {		
648		PRINTF("lm_inquire_device_list(%d)\n", number_of_devices);		
649		}		
650		#endif		
651		return (ret);		
652		}		
653		lm_inquire_device(device_number, attribute, value)		
654		long		
655		device_number,		
656		attribute,		
657		value,		
658		{		
659		long		
660		ret;		
661		{		
662		#ifdef DEBUG		
663		if (lm_debug[1]) {		
664		PRINTF("lm_inquire_device(%d, %d, %d)\n",		
665		device_number, attribute, value);		
666		}		
667		#endif		
668		clear_errors();		
669		if (!lm_sim_verified) {		
670		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
671		return (LM_ERROR);		
672		}		
673		value = 0;		
674		if (lm_system_selected == NULL) {		
675		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
676		return (LM_ERROR);		
677		}		
678		lm_choose_coas(lm_system_selected->id);		
679		lm_put_int(IMQ_DEVICE_CMD);		
680		lm_put_int(device_number);		
681		lm_put_int(attribute);		
682		if (!lm_end_put(lm_system_selected->id))		
683		return (LM_ERROR);		
684		if (lm_get_int() != IMQ_DEVICE_ANS) {		
685		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
686		lm_system_selected->name);		
687		return (LM_ERROR);		
688		}		
689		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
690		return (LM_ERROR);		
691		}		
692		value = lm_get_int();		
693		#ifdef DEBUG		
694		if (lm_debug[2]) {		
695		PRINTF("lm_inquire_device(%d, %d, %d)\n",		
696		device_number, attribute, value);		
697		}		
698		#endif		
699		return (ret);		
700		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 7/53
LINE #		SOURCE TEXT		
721		lm_inquire_dab(device_number, dab_number, attributes, string)		
722		long device_number;		
723		long dab_number;		
724		long attributes;		
725		char **string;		
726		{		
727		long ret;		
728		char *pro_str;		
729		#ifdef DEBUG		
730		if (lm_debug[1]) {		
731		DPRINTF(("lm_inquire_dab(%d, %d, %d, %d, %d)\n",		
732		device_number, dab_number, attributes, string));		
733		}		
734		#endif		
735		clear_errors();		
736		if (!lm_sim_verified) {		
737		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
738		return (LM_ERROR);		
739		}		
740		*string = NULL;		
741		if (lm_system_selected == NULL) {		
742		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
743		return (LM_ERROR);		
744		}		
745		lm_choose_conn(lm_system_selected->fd);		
746		lm_put_int(INQ_DAB_CMD);		
747		lm_put_int(device_number);		
748		lm_put_int(dab_number);		
749		lm_put_int(attributes);		
750		if (!lm_end_put(lm_system_selected->fd))		
751		return (LM_ERROR);		
752		if (lm_get_int() != INQ_DAB_ANS) {		
753		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
754		lm_system_selected->name);		
755		return (LM_ERROR);		
756		}		
757		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
758		return (LM_ERROR);		
759		pro_str = (char *) LM_GET_ADDR(lm_global_conn_ptr);		
760		(void) strcpy(lm_ind_string, pro_str);		
761		*string = lm_ind_string;		
762		#ifdef DEBUG		
763		if (lm_debug[2]) {		
764		DPRINTF(("lm_inquire_dab(%d, %d, %d, %d, %d)\n",		
765		*string));		
766		}		
767		#endif		
768		return (ret);		
769		}		
770		lm_inquire_dab_location(device_number, dab_number,		
771		number_of_locations, lanes, slots)		
772		long device_number;		
773		long dab_number;		
774		long number_of_locations;		
775		long lanes[];		
776		long slots[];		
777		{		
778		long ret;		
779		u_short i;		
780		#ifdef DEBUG		
781		if (lm_debug[1]) {		
782		DPRINTF(("lm_inquire_dab_location(%d, %d, %d, %d, %d)\n",		
783		device_number, dab_number,		
784		number_of_locations, lanes, slots));		
785		}		
786		#endif		
787		clear_errors();		
788		if (!lm_sim_verified) {		
789		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
790		return (LM_ERROR);		
791		}		
792		*number_of_locations = 0;		
793		if (lm_system_selected == NULL) {		
794		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
795		return (LM_ERROR);		
796		}		
797		lm_choose_conn(lm_system_selected->fd);		
798		lm_put_int(INQ_DAB_LOC_CMD);		
799		lm_put_int(device_number);		
800		lm_put_int(dab_number);		
801		if (!lm_end_put(lm_system_selected->fd))		
802		return (LM_ERROR);		
803		if (lm_get_int() != INQ_DAB_LOC_ANS) {		
804		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
805		lm_system_selected->name);		
806		return (LM_ERROR);		
807		}		
808		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
809		return (LM_ERROR);		
810		*number_of_locations = lm_get_int();		
811		for (i = 0; i < *number_of_locations; ++i) {		
812		lm_ida_lanes[i] = lm_get_int();		
813		lm_ida_slots[i] = lm_get_int();		
814		}		
815		*lanes = lm_ida_lanes;		
816		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 8/54
LINE #		SOURCE TEXT		
841		*slots = lm_ida_slots;		
842		}		
843		#ifdef DEBUG		
844		if (lm_debug[2]) {		
845		if (number_of_locations) {		
846		DPRINTF(("lm inquire dab location(XX, XX, %d, ",		
847		number_of_locations));		
848		for (i = 0; i < number_of_locations - 1; ++i) {		
849		DPRINTF(("(%d, %d), ", lm_ida_lanes[i], lm_ida_slots[i]));		
850		}		
851		DPRINTF(("(%d, %d)\n", lm_ida_lanes[i], lm_ida_slots[i]));		
852		}		
853		else {		
854		DPRINTF(("lm inquire dab location(XX, XX, %d)\n",		
855		number_of_locations));		
856		}		
857		}		
858		#endif		
859		return (ret);		
860		}		
861		}		
862		}		
863		lm_inquire_instance(instance_id, attribute, value)		
864		long instance_id;		
865		long attribute;		
866		long value;		
867		{		
868		INSTANCE_INFO *temp;		
869		long ret;		
870		u_short fault_count;		
871		u_short i;		
872		{		
873		#ifdef DEBUG		
874		if (lm_debug[1]) {		
875		DPRINTF(("lm inquire instance(%d, %d, %d)\n",		
876		instance_id, attribute, value));		
877		}		
878		#endif		
879		{		
880		clear_errors();		
881		if (!lm_sim_verified) {		
882		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
883		return (LM_ERROR);		
884		}		
885		value = 0;		
886		if (lm_specify_instance_id(instance_id) == LM_ERROR) {		
887		return (LM_ERROR);		
888		}		
889		if (attribute == LM_NUMBER_OF_VAULTS) {		
890		if (lm_instance_selected->assoc_inst_id == -1) {		
891		lm_queue_message(ERROR_MSG, "instance id: %d is a fault",		
892		instance_id);		
893		return (LM_ERROR);		
894		}		
895		fault_count = 0;		
896		for (i = 0; i < lm_inst_id_table_size; ++i) {		
897		temp = lm_inst_id_table[i];		
898		if (ISOCUS_INSTANCE(temp))		
899		continue;		
900		if (temp->assoc_inst_id == instance_id)		
901		++fault_count;		
902		}		
903		value = fault_count;		
904		#ifdef DEBUG		
905		if (lm_debug[2]) {		
906		DPRINTF(("lm inquire instance(XX, XX, %d)\n",		
907		value));		
908		}		
909		#endif		
910		{		
911		return (LM_SUCCESS);		
912		} else if (attribute == LM_ASSOCIATED_DEFINITION_ID) {		
913		value = lm_instance_selected->def_id;		
914		#ifdef DEBUG		
915		if (lm_debug[2]) {		
916		DPRINTF(("lm inquire instance(XX, XX, %d)\n",		
917		value));		
918		}		
919		#endif		
920		{		
921		return (LM_SUCCESS);		
922		}		
923		lm_choose_conn(lm_instance_selected->box_id);		
924		lm_put_int(INQ_INSTANCE_CMD);		
925		lm_put_short(lm_instance_selected->box_inst_id);		
926		lm_put_int(attribute);		
927		if (!lm_and_put(lm_instance_selected->box_id))		
928		return (LM_ERROR);		
929		if (lm_get_int() != INQ_INSTANCE_ANS) {		
930		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %d",		
931		lm_def_id_table[lm_instance_selected->def_id]->system_ptr->name);		
932		return (LM_ERROR);		
933		}		
934		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
935		return (LM_ERROR);		
936		value = lm_get_int();		
937		#ifdef DEBUG		
938		if (lm_debug[2]) {		
939		DPRINTF(("lm inquire instance(XX, XX, %d)\n",		
940		value));		
941		}		
942		#endif		
943		{		
944		return (ret);		
945		}		
946		lm_inquire_fault(fault_id, attribute, value)		
947		long fault_id;		
948		long attribute;		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 9/55
LINE #		SOURCE TEXT		
961	long	*value,		
962	{	long ret;		
963				
964	#ifdef DEBUG			
965	if (lm_debug[1]) {	DPRINTF(("lm_inquire_fault(%d, %d, %x)\n", fault_id, attributes, value));		
966				
967	DPRINTF(("lm_inquire_fault(%d, %d, %x)\n", fault_id, attributes, value));			
968	}			
969	#endif			
970	clear_errors();			
971				
972	if (!lm_sim_verified) {	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
973	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
974	return (LM_ERROR);			
975				
976	return (LM_ERROR);			
977	*value = 0;			
978	if (lm_specify_instance_id(fault_id) == LM_ERROR) {			
979	return (LM_ERROR);			
980				
981	if (attributes == LM_ASSOCIATED_INSTANCE_ID) {			
982	if (lm_instance_selected->assoc_inst_id == -1) {			
983	lm_queue_message(ERROR_MSG, "fault id: %d is an instance",			
984	fault_id);			
985	return (LM_ERROR);			
986				
987	*value = lm_instance_selected->assoc_inst_id;			
988				
989	#ifdef DEBUG			
990	if (lm_debug[2]) {	DPRINTF(("lm_inquire_fault(XX, XX, %d)\n", *value));		
991				
992	DPRINTF(("lm_inquire_fault(XX, XX, %d)\n", *value));			
993	}			
994	#endif			
995	return (LM_SUCCESS);			
996				
997	lm_choose_conn(lm_instance_selected->box_id);			
998	lm_put_inst(INQ_FAULT_CMD);			
999	lm_put_short(lm_instance_selected->box_inst_id);			
1000	lm_put_inst(attributes);			
1001	lm_put_inst(*value);			
1002				
1003	if (!lm_end_put(lm_instance_selected->box_id))			
1004	return (LM_ERROR);			
1005				
1006	if (lm_get_inst() != INQ_FAULT_ANS) {			
1007	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",			
1008	lm_def_id_table[lm_instance_selected->def_id]->system_ptr->name);			
1009	return (LM_ERROR);			
1010				
1011	if ((ret = lm_dequeue_errors()) == LM_ERROR)			
1012	return (LM_ERROR);			
1013				
1014	*value = lm_get_inst();			
1015				
1016	#ifdef DEBUG			
1017	if (lm_debug[2]) {	DPRINTF(("lm_inquire_fault(XX, XX, %d)\n", *value));		
1018				
1019	DPRINTF(("lm_inquire_fault(XX, XX, %d)\n", *value));			
1020	}			
1021	#endif			
1022	return (ret);			
1023				
1024	/*			
1025	*/			
1026	lm_select_modeler(modeler_name)			
1027	char	*modeler_name;		
1028	{			
1029				
1030	/*			
1031	* This function first search the system name in the system table. If found			
1032	* then set the system selected, if not found then try to initiate			
1033	* connection to that system and obtain the system info from that system and			
1034	* enter it into system table. The last element in the lm_system_table is			
1035	* used as a sentinel and it has to be always NULL.			
1036	*/			
1037				
1038	u_short	i;		
1039	short	skt;		
1040	u_short	error_count;		
1041	u_short	warning_count;		
1042				
1043	#ifdef DEBUG			
1044	if (lm_debug[1]) {	DPRINTF(("lm_select_modeler(%s)\n", modeler_name));		
1045				
1046	DPRINTF(("lm_select_modeler(%s)\n", modeler_name));			
1047	}			
1048	#endif			
1049	clear_errors();			
1050				
1051	if (!lm_sim_verified) {	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1052	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1053	return (LM_ERROR);			
1054				
1055	if (lm_verify_name(modeler_name, "modeler_name", MAX_NAME_LENGTH) == FAILURE)			
1056	return (LM_ERROR);			
1057				
1058	for (i = 0; lm_system_table[i] != NULL; ++i) {			
1059	if (strcmp(lm_system_table[i]->name, modeler_name) == 0) {			
1060	lm_system_selected = lm_system_table[i];			
1061	break;			
1062				
1063	}			
1064				
1065	if (lm_system_table[i] == NULL) {			
1066	lm_queue_message(WARNING_MSG, "modeler: %s is not in %s file",			
1067	modeler_name, HOSTFILENAME);			
1068				
1069	/* allow user to specify modeler not in HOSTFILENAME */			
1070	if (lm_enter_system(modeler_name, -1) == FAILURE)			
1071	return (LM_ERROR);			
1072				
1073	switch (lm_system_table[i]->state) {			
1074	case USED_FOR_RD_WR:			
1075	(void)lm_close_read_write_connection();			
1076	lm_system_table[i]->state = NOT_OPENED;			
1077	lm_system_table[i]->id = -1;			
1078	case NOT_OPENED:			
1079	if (lm_initconn(modeler_name, skt) == FAILURE) {			
1080	/* lm_system_table[i]->state = UNAVAIL; */			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	\$	DATE 5/23/89	PAGE # 10/56
				TIME 1:20:56 pm	
SOURCE TEXT					
LINE #					
1081	lm_system_selected = NULL;				
1082	return (LM_ERROR);				
1083	} else {				
1084	if (lm_send_begin_session_cmd((char)skt,				
1085	modeler_name) == FAILURE) {				
1086	lm_close_modeler(skt);				
1087	lm_system_selected = NULL;				
1088	return (LM_ERROR);				
1089	}				
1090	lm_system_table[i]-->state = USED;				
1091	lm_system_table[i]-->fd = skt;				
1092	lm_system_selected = lm_system_table[i];				
1093	}				
1094	break;				
1095	case UNAVAIL:				
1096	lm_queue_message(ERROR_MSG, "modeler: %s is unavailable",				
1097	modeler_name);				
1098	return (LM_ERROR);				
1099	case USED:				
1100	lm_system_selected = lm_system_table[i];				
1101	return (LM_SUCCESS);				
1102	default:				
1103	lm_queue_message(ERROR_MSG, "internal error: unknown modeler accessibility state");				
1104	return (LM_ERROR);				
1105	}				
1106	lm_message_types(&error_count, &warning_count);				
1107	}				
1108	if (error_count != 0)				
1109	return (LM_ERROR);				
1110	}				
1111	if (warning_count != 0)				
1112	return (LM_WARNING);				
1113	}				
1114	return (LM_SUCCESS);				
1115	}				
1116	lm_shutdown(seconds, flag)				
1117	long seconds;				
1118	long flag;				
1119	{				
1120	long ret;				
1121	u_long delay;				
1122	u_char i;				
1123	}				
1124	#ifdef DEBUG				
1125	if (lm_debug[1]) {				
1126	DPRINTF(("lm_shutdown(%d, %d)\n", seconds, flag));				
1127	}				
1128	#endif				
1129	{				
1130	clear_errors();				
1131	}				
1132	if (!lm_sim_verified) {				
1133	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");				
1134	return (LM_ERROR);				
1135	}				
1136	if (lm_system_selected == NULL) {				
1137	lm_queue_message(ERROR_MSG, "no modeler has been selected");				
1138	return (LM_ERROR);				
1139	}				
1140	lm_choose_conn(lm_system_selected->fd);				
1141	}				
1142	lm_put_int(SHUTDOWN_CMD);				
1143	lm_put_char(flag);				
1144	lm_put_int(seconds);				
1145	}				
1146	if (!lm_end_put(lm_system_selected->fd))				
1147	return (LM_ERROR);				
1148	}				
1149	if (lm_get_int() != SHUTDOWN_ANS) {				
1150	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",				
1151	lm_system_selected->name);				
1152	return (LM_ERROR);				
1153	}				
1154	if ((ret = lm_dequeue_errors()) == LM_ERROR)				
1155	return (LM_ERROR);				
1156	}				
1157	for (i = 0; lm_system_table[i] != NULL; ++i) {				
1158	if (lm_system_table[i] == lm_system_selected)				
1159	break;				
1160	}				
1161	}				
1162	if (lm_system_table[i] == NULL) {				
1163	lm_queue_message(ERROR_MSG, "internal error: system table corrupted");				
1164	return (FAILURE);				
1165	}				
1166	/* We want to wait for all users to complete */				
1167	if (flag == (u_long) LM_TRUE) {				
1168	lm_choose_conn(lm_system_selected->fd);				
1169	lm_put_int(ABORT_CMD);				
1170	}				
1171	if (!lm_end_put(lm_system_selected->fd))				
1172	return (LM_ERROR);				
1173	}				
1174	if (lm_get_int() != ABORT_ANS) {				
1175	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",				
1176	lm_system_selected->name);				
1177	return (LM_ERROR);				
1178	}				
1179	if ((ret = lm_dequeue_errors()) == LM_ERROR)				
1180	return (LM_ERROR);				
1181	}				
1182	}				
1183	(void) lm_close_read_write_connection();				
1184	lm_close_modeler(lm_system_table[i]-->fd);				
1185	lm_system_table[i]-->fd = -1;				
1186	lm_system_table[i]-->state = NOT_OPENED;				
1187	}				
1188	lm_system_selected = NULL;				
1189	}				
1190	delay = 4000000;				
1191	while (delay--);				
1192	}				
1193	return (ret);				
1194	}				
1195	}				
1196	}				
1197	}				
1198	}				
1199	}				
1200	}				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 11/57
LINE #		SOURCE TEXT		
1201		lm_test_network(total_count, loop_count)		
1202		long total_count,		
1203		long loop_count,		
1204		{		
1205		/* transmit "total_count" number of loops "loop_count" times */		
1206		{		
1207		u_long i,		
1208		u_long checksum,		
1209		u_long checksum2,		
1210		u_short loopno,		
1211		{		
1212		#ifdef DEBUG		
1213		if (lm_debug[1]) {		
1214		DPRINTF(("lm_test_network(%d, %d)\n", total_count, loop_count));		
1215		}		
1216		#endif		
1217		clear_errors();		
1218		if (!lm_sim_verified) {		
1219		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1220		return (LM_ERROR);		
1221		}		
1222		if (lm_system_selected == NULL) {		
1223		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
1224		return (LM_ERROR);		
1225		}		
1226		lm_choose_conn(lm_system_selected->fd),		
1227		lm_put_int(TEST_NETWORK_CMD),		
1228		lm_put_int(1); /* command */		
1229		lm_put_int(total_count);		
1230		checksum = 0;		
1231		for (i = 0; i < total_count; ++i) {		
1232		lm_put_int(i);		
1233		checksum += i;		
1234		}		
1235		lm_put_int(checksum);		
1236		DPRINTF(("checksum on command: %d\n", checksum));		
1237		if (!lm_and_put(lm_system_selected->fd))		
1238		return (LM_ERROR);		
1239		if (lm_get_int() != TEST_NETWORK_ANS) {		
1240		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
1241		lm_system_selected->name);		
1242		return (LM_ERROR);		
1243		}		
1244		if (lm_dequeue_errors() == LM_ERROR)		
1245		return (LM_ERROR);		
1246		total_count = lm_get_int();		
1247		DPRINTF(("total number is answer: %d\n", total_count));		
1248		checksum = 0;		
1249		for (i = 0; i < total_count; ++i) {		
1250		checksum += lm_get_int();		
1251		}		
1252		DPRINTF(("checksum on answer: %d\n", checksum));		
1253		if ((checksum2 = lm_get_int()) != checksum) {		
1254		lm_queue_message(ERROR_MSG, "internal error: checksum error on reply, exp: %08x got: %08x",		
1255		checksum2, checksum);		
1256		return (LM_ERROR);		
1257		}		
1258		--loop_count;		
1259		}		
1260		for (loopno = 0; loopno < loop_count; ++loopno) {		
1261		lm_choose_conn(lm_system_selected->fd);		
1262		{		
1263		lm_put_int(TEST_NETWORK_CMD);		
1264		lm_put_int(1); /* command */		
1265		lm_put_int(total_count);		
1266		checksum = 0;		
1267		for (i = 0; i < total_count; ++i) {		
1268		lm_put_int(i);		
1269		checksum += i;		
1270		}		
1271		lm_put_int(checksum);		
1272		if (!lm_and_put(lm_system_selected->fd))		
1273		return (LM_ERROR);		
1274		if (lm_get_int() != TEST_NETWORK_ANS) {		
1275		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
1276		lm_system_selected->name);		
1277		return (LM_ERROR);		
1278		}		
1279		if (lm_dequeue_errors() == LM_ERROR)		
1280		return (LM_ERROR);		
1281		total_count = lm_get_int();		
1282		checksum = 0;		
1283		for (i = 0; i < total_count; ++i) {		
1284		checksum += lm_get_int();		
1285		}		
1286		if ((checksum2 = lm_get_int()) != checksum) {		
1287		lm_queue_message(ERROR_MSG, "internal error: checksum error on reply, exp: %08x got: %08x",		
1288		checksum2, checksum);		
1289		return (LM_ERROR);		
1290		}		
1291		return (LM_SUCCESS);		
1292		}		
1293		lm_label_dab(lane_number, slot_number,		
1294		dab_type, device_name,		
1295		maker, maker_revision,		
1296		manufacturer, revision,		
1297		lane_high, slots_wide,		
1298		segment_number)		
1299		long lane_number,		
1300		long slot_number,		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 12/58
LINE #	SOURCE TEXT			
1321	char	"dab_type,		
1322	char	"device_name,		
1323	char	"maker,		
1324	char	"maker_revision,		
1325	char	"manufacturer,		
1326	char	"revision,		
1327	long	lane_high,		
1328	long	slot_wide,		
1329	long	segment_number,		
1330	{			
1331	long	ret,		
1332	u_char	i,		
1333	}			
1334	#ifdef DEBUG			
1335	if (lm_debug[1]) {			
1336	PRINTF(("lm_label dab(%d, %d, %s, %s, %s, %s, %s, %d, %d, %d)\n",			
1337	lane_number, slot_number,			
1338	dab_type, device_name,			
1339	maker, maker_revision,			
1340	manufacturer, revision,			
1341	lane_high, slot_wide,			
1342	segment_number));			
1343	}			
1344	#endif			
1345				
1346	clear_errors();			
1347	if (!lm_sim_verified) {			
1348	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1349	return (LM_ERROR);			
1350	}			
1351	if (lm_system_selected == NULL) {			
1352	lm_queue_message(ERROR_MSG, "no modeler has been selected");			
1353	return (LM_ERROR);			
1354	}			
1355	lm_choose_conn(lm_system_selected->fd);			
1356				
1357	lm_put_int(LABEL_DAB_CMD);			
1358				
1359	lm_put_char(lane_number);			
1360				
1361	lm_put_char(slot_number);			
1362				
1363	if (dab_type == NULL) {			
1364	lm_queue_message(ERROR_MSG, "dab_type is NULL");			
1365	return (LM_ERROR);			
1366	}			
1367	if (strlen(dab_type) == 0) {			
1368	lm_queue_message(ERROR_MSG, "dab_type is an empty string");			
1369	return (LM_ERROR);			
1370	}			
1371	if (lm_isascii_string(dab_type) == FALSE) {			
1372	lm_queue_message(ERROR_MSG, "internal simulator error: dab_type is not an ascii string");			
1373	return (LM_ERROR);			
1374	}			
1375	/* dab_type */			
1376	for (i = 0; ((i < 127) && (dab_type[i] != '\0')); ++i)			
1377	lm_put_char(dab_type[i]);			
1378	lm_put_char('\0');			
1379				
1380	if (device_name == NULL) {			
1381	lm_queue_message(ERROR_MSG, "device_name is NULL");			
1382	return (LM_ERROR);			
1383	}			
1384	if (strlen(device_name) == 0) {			
1385	lm_queue_message(ERROR_MSG, "device_name is an empty string");			
1386	return (LM_ERROR);			
1387	}			
1388	if (lm_isascii_string(device_name) == FALSE) {			
1389	lm_queue_message(ERROR_MSG, "internal simulator error: device_name is not an ascii string");			
1390	return (LM_ERROR);			
1391	}			
1392	/* device_name */			
1393	for (i = 0; ((i < 127) && (device_name[i] != '\0')); ++i)			
1394	lm_put_char(device_name[i]);			
1395	lm_put_char('\0');			
1396				
1397	if (maker == NULL) {			
1398	lm_queue_message(ERROR_MSG, "maker is NULL");			
1399	return (LM_ERROR);			
1400	}			
1401	if (strlen(maker) == 0) {			
1402	lm_queue_message(ERROR_MSG, "maker is an empty string");			
1403	return (LM_ERROR);			
1404	}			
1405	if (lm_isascii_string(maker) == FALSE) {			
1406	lm_queue_message(ERROR_MSG, "internal simulator error: maker is not an ascii string");			
1407	return (LM_ERROR);			
1408	}			
1409	/* maker */			
1410	for (i = 0; ((i < 127) && (maker[i] != '\0')); ++i)			
1411	lm_put_char(maker[i]);			
1412	lm_put_char('\0');			
1413				
1414	if (maker_revision == NULL) {			
1415	lm_queue_message(ERROR_MSG, "maker_revision is NULL");			
1416	return (LM_ERROR);			
1417	}			
1418	if (strlen(maker_revision) == 0) {			
1419	lm_queue_message(ERROR_MSG, "maker_revision is an empty string");			
1420	return (LM_ERROR);			
1421	}			
1422	if (lm_isascii_string(maker_revision) == FALSE) {			
1423	lm_queue_message(ERROR_MSG, "internal simulator error: maker_revision is not an ascii string");			
1424	return (LM_ERROR);			
1425	}			
1426	/* maker_revision */			
1427	for (i = 0; ((i < 127) && (maker_revision[i] != '\0')); ++i)			
1428	lm_put_char(maker_revision[i]);			
1429	lm_put_char('\0');			
1430				
1431	if (manufacturer == NULL) {			
1432	lm_queue_message(ERROR_MSG, "manufacturer is NULL");			
1433	return (LM_ERROR);			
1434	}			
1435	if (strlen(manufacturer) == 0) {			
1436	lm_queue_message(ERROR_MSG, "manufacturer is an empty string");			
1437	return (LM_ERROR);			
1438	}			
1439	if (lm_isascii_string(manufacturer) == FALSE) {			
1440	lm_queue_message(ERROR_MSG, "internal simulator error: manufacturer is not an ascii string");			
1441	return (LM_ERROR);			
1442	}			
1443	/* manufacturer */			
1444	for (i = 0; ((i < 127) && (manufacturer[i] != '\0')); ++i)			
1445	lm_put_char(manufacturer[i]);			
1446	lm_put_char('\0');			
1447				
1448	if (lane_high == NULL) {			
1449	lm_queue_message(ERROR_MSG, "lane_high is NULL");			
1450	return (LM_ERROR);			
1451	}			
1452	if (slot_wide == NULL) {			
1453	lm_queue_message(ERROR_MSG, "slot_wide is NULL");			
1454	return (LM_ERROR);			
1455	}			
1456	if (segment_number == NULL) {			
1457	lm_queue_message(ERROR_MSG, "segment_number is NULL");			
1458	return (LM_ERROR);			
1459	}			
1460	if (strlen(segment_number) == 0) {			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 13/59
LINE #		SOURCE TEXT		
1441		lm_queue_message(ERROR_MSG, "manufacturer is an empty string");		
1442		return (LM_ERROR);		
1443		{		
1444		if (lm_isascii_string(manufacturer) == FALSE) {		
1445		lm_queue_message(ERROR_MSG, "internal simulator error: manufacturer is not an ascii string");		
1446		return (LM_ERROR);		
1447		}		
1448		/* manufacturer */		
1449		for (i = 0; ((i < 127) && (manufacturer[i] != '\0')); ++i)		
1450		lm_put_char(manufacturer[i]);		
1451		lm_put_char('\0');		
1452		{		
1453		if (revision == NULL) {		
1454		lm_queue_message(ERROR_MSG, "revision is NULL");		
1455		return (LM_ERROR);		
1456		}		
1457		if (strlen(revision) == 0) {		
1458		lm_queue_message(ERROR_MSG, "revision is an empty string");		
1459		return (LM_ERROR);		
1460		}		
1461		if (lm_isascii_string(revision) == FALSE) {		
1462		lm_queue_message(ERROR_MSG, "internal simulator error: revision is not an ascii string");		
1463		return (LM_ERROR);		
1464		}		
1465		/* revision */		
1466		for (i = 0; ((i < 127) && (revision[i] != '\0')); ++i)		
1467		lm_put_char(revision[i]);		
1468		lm_put_char('\0');		
1469		{		
1470		lm_put_char(lanes_high);		
1471		lm_put_char(slots_wide);		
1472		lm_put_char(segment_number);		
1473		{		
1474		if (!lm_end_put(lm_system_selected->fd))		
1475		return (LM_ERROR);		
1476		{		
1477		if (lm_get_int() != LABEL_DAB_ANS) {		
1478		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
1479		lm_system_selected->name);		
1480		return (LM_ERROR);		
1481		}		
1482		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
1483		return (LM_ERROR);		
1484		{		
1485		return (ret);		
1486		}		
1487		{		
1488		lm_eval_control(instance_or_fault_id, attribute, value)		
1489		long instance_or_fault_id,		
1490		long attribute,		
1491		unsigned long value;		
1492		{		
1493		long ret;		
1494		{		
1495		#ifdef DEBUG		
1496		if (lm_debug[1]) {		
1497		DPRINTF(("lm_eval_control(%d, %d, %d)\n",		
1498		instance_or_fault_id, attribute, value));		
1499		}		
1500		#endif		
1501		{		
1502		clear_errors();		
1503		{		
1504		if (!lm_sim_verified) {		
1505		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1506		return (LM_ERROR);		
1507		{		
1508		if (lm_specify_instance_id(instance_or_fault_id) == LM_ERROR) {		
1509		return (LM_ERROR);		
1510		{		
1511		lm_choose_conn(lm_instance_selected->box_id);		
1512		lm_put_int(EVAL_CONTROL_CMD);		
1513		lm_put_short(lm_instance_selected->box_inst_id);		
1514		lm_put_int(attribute);		
1515		lm_put_int(value);		
1516		{		
1517		if (!lm_end_put(lm_instance_selected->box_id))		
1518		return (LM_ERROR);		
1519		{		
1520		if (lm_get_int() != EVAL_CONTROL_ANS) {		
1521		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
1522		lm_def_id_table[lm_instance_selected->def_id]->system_ptr->name);		
1523		return (LM_ERROR);		
1524		{		
1525		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
1526		return (LM_ERROR);		
1527		{		
1528		return (ret);		
1529		{		
1530		{		
1531		lm_reboot(seconds, flag)		
1532		long seconds,		
1533		long flag;		
1534		{		
1535		long ret;		
1536		u_long delay;		
1537		u_char i;		
1538		{		
1539		#ifdef DEBUG		
1540		if (lm_debug[1]) {		
1541		DPRINTF(("lm_reboot(%d, %d)\n", seconds, flag));		
1542		}		
1543		#endif		
1544		{		
1545		clear_errors();		
1546		{		
1547		if (!lm_sim_verified) {		
1548		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1549		return (LM_ERROR);		
1550		{		
1551		if (lm_system_selected == NULL) {		
1552		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
1553		return (LM_ERROR);		
1554		{		
1555		lm_choose_conn(lm_system_selected->fd);		
1556		{		
1557		lm_put_int(REBOOT_CMD);		
1558		lm_put_char(flag);		
1559		lm_put_int(seconds);		
1560		{		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 14/60
LINE #	SOURCE TEXT			
551	if (!lm_end_put(lm_system_selected->fd))			
552	return (LM_ERROR);			
553	if (lm_get_int() != SUCCESS_MSG) {			
554	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",			
555	lm_system_selected->name);			
556	return (LM_ERROR);			
557	}			
558	if ((ret = lm_dequeue_errors()) == LM_ERROR)			
559	return (LM_ERROR);			
560	for (i = 0; lm_system_table[i] != NULL; ++i) {			
561	if (lm_system_table[i] == lm_system_selected)			
562	break;			
563	}			
564	if (lm_system_table[i] == NULL) {			
565	lm_queue_message(ERROR_MSG, "internal error: system table corrupted");			
566	return (FAILURE);			
567	}			
568	/* We want to wait for all users to complete */			
569	if (flag == (u_long) LM_DONE) {			
570	lm_choose_conn(lm_system_selected->fd);			
571	lm_put_int(ABORT_CMD);			
572	if (!lm_end_put(lm_system_selected->fd))			
573	return (LM_ERROR);			
574	if (lm_get_int() != ABORT_MSG) {			
575	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",			
576	lm_system_selected->name);			
577	return (LM_ERROR);			
578	}			
579	if ((ret = lm_dequeue_errors()) == LM_ERROR)			
580	return (LM_ERROR);			
581	}			
582	(void) lm_close_read_write_connection();			
583	lm_close_modeler(lm_system_table[i]->fd);			
584	lm_system_table[i]->fd = -1;			
585	lm_system_table[i]->state = NOT_OPENED;			
586	lm_system_selected = NULL;			
587	delay = 1000000;			
588	while (delay--;			
589	return (ret);			
590	}			
591	lm_inquire_device_pins(definition_id, pin_type, pin_name, pin_number_str, pin_alias, pin_attribute)			
592	long definition_id;			
593	long pin_type;			
594	char **pin_name;			
595	char **pin_number_str;			
596	char **pin_alias;			
597	long *pin_attribute;			
598	{			
599	short pin_number;			
600	u_char ptype;			
601	{			
602	#ifdef DEBUG			
603	if (lm_debug[1]) {			
604	PRINTF("lm_inquire_device_pins(%d, %d, %s, %s, %s, %s)\n",			
605	definition_id, pin_type, pin_name, pin_number_str,			
606	pin_alias, pin_attribute);			
607	}			
608	#endif			
609	clear_errors();			
610	if (!lm_sim_verified) {			
611	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
612	return (LM_ERROR);			
613	}			
614	if ((definition_id != lm_last_definition_id_specified)			
615	(definition_id == -1)) {			
616	if ((definition_id < 0) (definition_id >= lm_def_id_avail)) {			
617	lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",			
618	definition_id);			
619	return (LM_ERROR);			
620	}			
621	if (lm_def_id_table[definition_id] == NULL) {			
622	lm_queue_message(ERROR_MSG, "internal simulator error: invalid definition id: %d specified",			
623	definition_id);			
624	return (LM_ERROR);			
625	}			
626	lm_last_definition_id_specified = definition_id;			
627	lm_definition_selected = lm_def_id_table[definition_id];			
628	}			
629	switch (pin_type) {			
630	case LM_INPUT:			
631	pin_number = lm_definition_selected->input_pin_number_returned;			
632	ptype = IN;			
633	break;			
634	case LM_OUTPUT:			
635	pin_number = lm_definition_selected->output_pin_number_returned;			
636	ptype = OUT;			
637	break;			
638	case LM_IO:			
639	pin_number = lm_definition_selected->io_pin_number_returned;			
640	ptype = IO;			
641	break;			
642	case LM_POWER:			
643	pin_number = lm_definition_selected->power_pin_number_returned;			
644	ptype = POWER;			
645	break;			
646	case LM_GROUND:			
647	pin_number = lm_definition_selected->ground_pin_number_returned;			
648	ptype = GROUND;			
649	break;			
650	case LM_NO_CONNECT:			
651	pin_number = lm_definition_selected->nc_pin_number_returned;			
652	ptype = NC;			
653	break;			
654	default:			
655	lm_queue_message(ERROR_MSG, "internal simulator error: invalid pin_type: %d specified",			
656	pin_type);			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 15/61
LINE #		SOURCE TEXT		
1681		return (LM_ERROR);		
1682		}		
1683		for (++pin_number,		
1684		pin_number < lm_definition_selected->dab_pid_table_size, ++pin_number) {		
1685		if (lm_definition_selected->dab_pid_table[pin_number].pin_type == ptype) {		
1686		break;		
1687		}		
1688		}		
1689		}		
1690		if (pin_number == lm_definition_selected->dab_pid_table_size)		
1691		pin_number = -1;		
1692		switch (pin_type) {		
1693		case LM_INPUT:		
1694		lm_definition_selected->input_pin_number_returned = pin_number;		
1695		break;		
1696		case LM_OUTPUT:		
1697		lm_definition_selected->output_pin_number_returned = pin_number;		
1698		break;		
1699		case LM_IO:		
1700		lm_definition_selected->io_pin_number_returned = pin_number;		
1701		break;		
1702		case LM_POWER:		
1703		lm_definition_selected->power_pin_number_returned = pin_number;		
1704		break;		
1705		case LM_GROUND:		
1706		lm_definition_selected->ground_pin_number_returned = pin_number;		
1707		break;		
1708		case LM_NO_CONNECT:		
1709		lm_definition_selected->ac_pin_number_returned = pin_number;		
1710		break;		
1711		default:		
1712		break;		
1713		}		
1714		if (pin_number == -1) {		
1715		pin_name = LM_NO_MORE_PINS;		
1716		pin_number_str = "";		
1717		pin_attribute = 0;		
1718		pin_alias = "";		
1719		}		
1720		#ifdef DEBUG		
1721		if (lm_debug[2]) {		
1722		DPRINTF(("lm_inquire_device_pins(XX, XX, LM_NO_MORE_PINS, (null), (null), %d)\n",		
1723		pin_attribute));		
1724		}		
1725		#endif		
1726		return (LM_SUCCESS);		
1727		}		
1728		switch (lm_definition_selected->dab_pid_table[pin_number].pin_class) {		
1729		case DATA:		
1730		pin_attribute = LM_DATA_PIN;		
1731		break;		
1732		case EVAL:		
1733		pin_attribute = LM_EVAL_PIN;		
1734		break;		
1735		case STORE:		
1736		pin_attribute = LM_STORE_PIN;		
1737		break;		
1738		case EDGE_RISE:		
1739		pin_attribute = LM_EDGE_RISE_PIN;		
1740		break;		
1741		case EDGE_FALL:		
1742		pin_attribute = LM_EDGE_FALL_PIN;		
1743		break;		
1744		default:		
1745		break;		
1746		}		
1747		(void) strcpy(lm_idp_string,		
1748		lm_definition_selected->dab_pid_table[pin_number].name);		
1749		pin_name = lm_idp_string;		
1750		(void) strcpy(lm_idp_string2,		
1751		lm_definition_selected->dab_pid_table[pin_number].pin_number_str);		
1752		pin_number_str = lm_idp_string2;		
1753		if (lm_definition_selected->dab_pid_table[pin_number].pin_alias == NULL) {		
1754		pin_alias = "";		
1755		}		
1756		else {		
1757		(void) strcpy(lm_idp_string3,		
1758		lm_definition_selected->dab_pid_table[pin_number].pin_alias);		
1759		pin_alias = lm_idp_string3;		
1760		}		
1761		#ifdef DEBUG		
1762		if (lm_debug[2]) {		
1763		DPRINTF(("lm_inquire_device_pins(XX, XX, %s, %s, %s, %d)\n",		
1764		pin_name, pin_number_str, pin_alias, pin_attribute));		
1765		}		
1766		#endif		
1767		return (LM_SUCCESS);		
1768		}		
1769		lm_check_shell_software_f(filename)		
1770		char filename;		
1771		{		
1772		char device_name;		
1773		char modeler_name;		
1774		char output;		
1775		u_long n;		
1776		u_long buf_size;		
1777		short id;		
1778		short skt;		
1779		u_short device_usage;		
1780		u_short error_count;		
1781		u_short warning_count;		
1782		u_char i;		
1783		u_char opened_new_modeler = FALSE;		
1784		char full_filename[MAX_STRING_LENGTH];		
1785		#ifdef DEBUG		
1786		if (lm_debug[1]) {		
1787		DPRINTF(("lm_check_shell_software_f(%s)\n", filename));		
1788		}		
1789		#endif		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 16/62
LINE #	SOURCE TEXT			
1801	clear_errors();			
1802	if (!lm_verified) {			
1803	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
1804	return (LM_ERROR);			
1805	}			
1806	if (lm_verify_name(filename, "filename", MAX_STRING_LENGTH) == FAILURE)			
1807	return (LM_ERROR);			
1808	if (lm_resolve_library_file("LM_LIB", filename,			
1809	full_filename, (long) MAX_STRING_LENGTH) == FAILURE) {			
1810	return (LM_ERROR);			
1811	}			
1812	if (lm_check_file_type(full_filename) == FAILURE) {			
1813	return (LM_ERROR);			
1814	}			
1815	/* BAD/STYS compatibility change to hardcode O_RDONLY to equal 0. */			
1816	if ((fd = open(full_filename, 0)) == -1) {			
1817	lm_queue_message(ERROR_MSG, "cannot open shell software file: %s",			
1818	full_filename);			
1819	return (LM_ERROR);			
1820	}			
1821	lm_thuf_ptr = lm_temp_buffer;			
1822	buf_size = lm_max_thuf_addr - lm_temp_buffer - 1;			
1823	while ((n = (u_long) read(fd, lm_thuf_ptr, (int) buf_size)) > 0) {			
1824	lm_thuf_ptr += n;			
1825	buf_size -= n;			
1826	if (buf_size == 0) {			
1827	if (lm_extend_thuf() == FAILURE)			
1828	return (LM_ERROR);			
1829	buf_size = lm_max_thuf_addr - lm_thuf_ptr - 1;			
1830	}			
1831	*lm_thuf_ptr++ = '\0';			
1832	if (close(fd) != 0) {			
1833	lm_queue_message(ERROR_MSG, "cannot close file: %s",			
1834	full_filename);			
1835	return (LM_ERROR);			
1836	}			
1837	/* Run the NBSL front-end processor (lexical_pass(), lexical_pass()). Returns			
1838	*NULL if there are lexical errors.			
1839	*/			
1840	lm_init_vars("NBSL");			
1841	outbuf = lm_lexical_pass(full_filename, lm_temp_buffer, &device_name,			
1842	&modular_name, &device_usage);			
1843	if (outbuf == NULL) {			
1844	/* error encountered when parsing the DADSH */			
1845	return (LM_ERROR);			
1846	}			
1847	if (device_name == NULL) {			
1848	lm_queue_message(ERROR_MSG, "no device_name statement in shell software: %s",			
1849	full_filename);			
1850	return (LM_ERROR);			
1851	}			
1852	for (i = 0; lm_system_table[i] != NULL; ++i) {			
1853	if (lm_system_table[i] == USED)			
1854	break;			
1855	}			
1856	if (lm_system_table[i] == NULL) {			
1857	for (i = 0; lm_system_table[i] != NULL; ++i) {			
1858	if (lm_system_table[i] == NOT_OPENED)			
1859	continue;			
1860	if (lm_initconn(lm_system_table[i] == NAME, &kt) == FAILURE) {			
1861	/* lm_system_table[i] == NAME, &kt == FAILURE */			
1862	lm_system_table[i] == fd = -1;			
1863	} else {			
1864	if (lm_read_begin_session_cmd((char)kt,			
1865	lm_system_table[i] == NAME) == FAILURE) {			
1866	lm_close_modeler(&kt);			
1867	} else {			
1868	opened_new_modeler = TRUE;			
1869	lm_system_table[i] == state = USED;			
1870	lm_system_table[i] == fd = &kt;			
1871	break;			
1872	}			
1873	}			
1874	/*			
1875	* Clear the errors from all the modeler v. talked to when trying to find			
1876	* a good modeler to check shell software.			
1877	*/			
1878	clear_errors();			
1879	if (lm_system_table[i] == NULL) {			
1880	lm_queue_message(ERROR_MSG, "cannot find any modeler to check shell software");			
1881	return (LM_ERROR);			
1882	}			
1883	(void) lm_read_def2(lm_system_table[i], outbuf);			
1884	/* Free the buffer allocated by NBSL front-end processor */			
1885	free(outbuf);			
1886	if (opened_new_modeler == TRUE) {			
1887	lm_choose_conn(lm_system_table[i] == fd);			
1888	lm_put_int(ABORT_CMD);			
1889	(void) lm_end_put(lm_system_table[i] == fd);			
1890	lm_close_modeler(lm_system_table[i] == fd);			
1891	lm_system_table[i] == fd = -1;			
1892	lm_system_table[i] == state = NOT_OPENED;			
1893	}			
1894	lm_message_types(&error_count, &warning_count);			
1895	if (error_count != 0)			
1896	return (LM_ERROR);			
1897	if (warning_count != 0)			
1898	return (LM_WARNING);			
1899				
1900				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89	PAGE # 17/63
LINE #		SOURCE TEXT		
1921		return (LM_SUCCESS);		
1922		}		
1923		lm_check_shell_software_s(string)		
1924		char		
1925		{		
1926		char		
1927		char		
1928		char		
1929		char		
1930		char		
1931		short		
1932		u_short		
1933		u_short		
1934		u_short		
1935		u_char		
1936		u_char		
1937		{		
1938		#ifdef DEBUG		
1939		if (lm_debug[1]) {		
1940		DPRINTF(("lm_check_shell_software_s(%s)\n", string));		
1941		}		
1942		#endif		
1943		{		
1944		clear_errors();		
1945		if (!lm_sis_verified) {		
1946		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
1947		return (LM_ERROR);		
1948		}		
1949		if (lm_first_call == TRUE) {		
1950		lm_tbuf_ptr = lm_temp_buffer;		
1951		lm_first_call = FALSE;		
1952		}		
1953		if (string != LM_END_OF_TEXT) {		
1954		str = string;		
1955		if (lm_tbuf_ptr_addr - lm_tbuf_ptr - 1 < strlen(str) + 1)		
1956		if (lm_extend_tbuf() == FAILURE)		
1957		return (LM_ERROR);		
1958		for (; (lm_tbuf_ptr++ = str++) != '\n');		
1959		return (LM_SUCCESS);		
1960		}		
1961		lm_tbuf_ptr = '\0';		
1962		lm_first_call = TRUE;		
1963		}		
1964		/*		
1965		* Run the EBNL front-end processor (lexical_pass()). lexical_pass() returns		
1966		* NULL if there are lexical errors.		
1967		*/		
1968		lm_init_vars("EBNL");		
1969		outbuf = lm_lexical_pass("NULLNAME", lm_temp_buffer, &device_name,		
1970		&modeler_name, &device_usage);		
1971		}		
1972		if (outbuf == NULL) {		
1973		/* error encountered when parsing the DANDER */		
1974		return (LM_ERROR);		
1975		}		
1976		if (device_name == NULL) {		
1977		lm_queue_message(ERROR_MSG, "no device_name statement in Shell Software: 43");		
1978		return (LM_ERROR);		
1979		}		
1980		for (i = 0; lm_system_table[i] != NULL; ++i) {		
1981		if (lm_system_table[i] == USED)		
1982		break;		
1983		}		
1984		if (lm_system_table[i] == NULL) {		
1985		for (i = 0; lm_system_table[i] != NULL; ++i) {		
1986		if (lm_system_table[i] == NOT_OPENED)		
1987		continue;		
1988		if (lm_initconn(lm_system_table[i] == NAME, &akt) == FAILURE) {		
1989		/* lm_system_table[i] == NAME == UNAVAILABLE */		
1990		lm_system_table[i] == -1;		
1991		} else {		
1992		if (lm_send_begin_session_cmd((char)akt,		
1993		lm_system_table[i] == NAME) == FAILURE) {		
1994		lm_close_modeler(akt);		
1995		} else {		
1996		opened_new_modeler = TRUE;		
1997		lm_system_table[i] == STATE = USED;		
1998		lm_system_table[i] == ID = akt;		
1999		break;		
2000		}		
2001		}		
2002		}		
2003		}		
2004		}		
2005		}		
2006		}		
2007		if (lm_system_table[i] == NULL) {		
2008		lm_queue_message(ERROR_MSG, "cannot find any modeler to check Shell Software");		
2009		return (LM_ERROR);		
2010		}		
2011		}		
2012		(void) lm_send_def2(lm_system_table[i], outbuf);		
2013		}		
2014		/* Free the buffer allocated by EBNL front-end processor */		
2015		free(outbuf);		
2016		}		
2017		if (opened_new_modeler == TRUE) {		
2018		lm_choose_conn(lm_system_table[i] == ID);		
2019		lm_put_int(ABORT_CMD);		
2020		(void) lm_send_put(lm_system_table[i] == ID);		
2021		}		
2022		lm_close_modeler(lm_system_table[i] == ID);		
2023		lm_system_table[i] == ID = -1;		
2024		lm_system_table[i] == STATE = NOT_OPENED;		
2025		}		
2026		lm_message_types(&error_count, &warning_count);		
2027		}		
2028		if (error_count != 0)		
2029		return (LM_ERROR);		
2030		}		
2031		if (warning_count != 0)		
2032		return (LM_WARNING);		
2033		}		
2034		return (LM_SUCCESS);		
2035		}		
2036		}		
2037		}		
2038		}		
2039		}		
2040		lm_lock_modeler()		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 18/64
LINE #	SOURCE TEXT			
2041	{			
2042	u_char 1;			
2043	char c;			
2044	}			
2045	#ifdef DEBUG			
2046	if (!lm_debug[1]) {			
2047	DPRINTF(("lm_lock_modelar()\n"));			
2048	}			
2049	#endif			
2050	clear_errors();			
2051	if (!lm_sim_verified) {			
2052	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
2053	return (LM_ERROR);			
2054	}			
2055	if (lm_system_selected == NULL) {			
2056	lm_queue_message(ERROR_MSG, "no modelar has been selected");			
2057	return (LM_ERROR);			
2058	}			
2059	lm_choose_conn(lm_system_selected->fd);			
2060	lm_put_int(TEST_NETWORK_CMD);			
2061	lm_put_int(2); /* command */			
2062	lm_put_int(1000); /* subcommand */			
2063	for (i = 0; (c = lm_gbl_username[i]) != '\0'; ++i)			
2064	lm_put_char(c);			
2065	lm_put_char('\0');			
2066	if (!lm_end_put(lm_system_selected->fd))			
2067	return (LM_ERROR);			
2068	if (lm_get_int() != TEST_NETWORK_ANS) {			
2069	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modelar: %s",			
2070	lm_system_selected->name);			
2071	return (LM_ERROR);			
2072	}			
2073	return (lm_dequeue_errors());			
2074	}			
2075	lm_unlock_modelar()			
2076	{			
2077	#ifdef DEBUG			
2078	if (!lm_debug[1]) {			
2079	DPRINTF(("lm_unlock_modelar()\n"));			
2080	}			
2081	#endif			
2082	clear_errors();			
2083	if (!lm_sim_verified) {			
2084	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
2085	return (LM_ERROR);			
2086	}			
2087	if (lm_system_selected == NULL) {			
2088	lm_queue_message(ERROR_MSG, "no modelar has been selected");			
2089	return (LM_ERROR);			
2090	}			
2091	lm_choose_conn(lm_system_selected->fd);			
2092	lm_put_int(TEST_NETWORK_CMD);			
2093	lm_put_int(2); /* command */			
2094	lm_put_int(2000); /* subcommand */			
2095	if (!lm_end_put(lm_system_selected->fd))			
2096	return (LM_ERROR);			
2097	if (lm_get_int() != TEST_NETWORK_ANS) {			
2098	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modelar: %s",			
2099	lm_system_selected->name);			
2100	return (LM_ERROR);			
2101	}			
2102	return (lm_dequeue_errors());			
2103	}			
2104	lm_abort_user(user_number)			
2105	{			
2106	u_long user_number;			
2107	{			
2108	#ifdef DEBUG			
2109	if (!lm_debug[1]) {			
2110	DPRINTF(("lm_abort_user(%d)\n", user_number));			
2111	}			
2112	#endif			
2113	clear_errors();			
2114	if (!lm_sim_verified) {			
2115	lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");			
2116	return (LM_ERROR);			
2117	}			
2118	if (lm_system_selected == NULL) {			
2119	lm_queue_message(ERROR_MSG, "no modelar has been selected");			
2120	return (LM_ERROR);			
2121	}			
2122	lm_choose_conn(lm_system_selected->fd);			
2123	lm_put_int(TEST_NETWORK_CMD);			
2124	lm_put_int(3); /* command */			
2125	lm_put_int(user_number);			
2126	if (!lm_end_put(lm_system_selected->fd))			
2127	return (LM_ERROR);			
2128	if (lm_get_int() != TEST_NETWORK_ANS) {			
2129	lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modelar: %s",			
2130	lm_system_selected->name);			
2131	return (LM_ERROR);			
2132	}			
2133	return (lm_dequeue_errors());			
2134	}			
2135	}			
2136	}			
2137	}			
2138	}			
2139	}			
2140	}			
2141	}			
2142	}			
2143	}			
2144	}			
2145	}			
2146	}			
2147	}			
2148	}			
2149	}			
2150	}			
2151	}			
2152	}			
2153	}			
2154	}			
2155	}			
2156	}			
2157	}			
2158	}			
2159	}			
2160	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/sfi2.c	DATE 5/23/89 TIME 1:20:56 pm	PAGE # 19/65
LINE #		SOURCE TEXT		
2161		lm_clear_profile()		
2162		{		
2163		#ifdef DEBUG		
2164		if (lm_debug(1)) {		
2165		DPRINTF(("lm_clear_profile()\n"));		
2166		}		
2167		#endif		
2168		{		
2169		clear_errors();		
2170		if (!lm_sim_verified) {		
2171		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
2172		return (LM_ERROR);		
2173		}		
2174		if (lm_system_selected == NULL) {		
2175		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
2176		return (LM_ERROR);		
2177		}		
2178		lm_choose_coas(lm_system_selected->id);		
2179		{		
2180		lm_put_int(TEST_NETWORK_CMD);		
2181		lm_put_int(4); /* command */		
2182		if (!lm_end_put(lm_system_selected->id))		
2183		return (LM_ERROR);		
2184		if (lm_get_int() != TEST_NETWORK_ANS) {		
2185		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
2186		lm_system_selected->name);		
2187		return (LM_ERROR);		
2188		}		
2189		return (lm_dequeue_errors());		
2190		}		
2191		lm_dump_profile(filename)		
2192		{		
2193		FILE		
2194		long		
2195		long		
2196		long		
2197		long		
2198		long		
2199		count;		
2200		#ifdef DEBUG		
2201		if (lm_debug(1)) {		
2202		DPRINTF(("lm_dump_profile(%s)\n", filename));		
2203		}		
2204		#endif		
2205		{		
2206		clear_errors();		
2207		if (!lm_sim_verified) {		
2208		lm_queue_message(ERROR_MSG, "internal simulator error: lm_begin_modeling_session() has not yet been called");		
2209		return (LM_ERROR);		
2210		}		
2211		if (lm_system_selected == NULL) {		
2212		lm_queue_message(ERROR_MSG, "no modeler has been selected");		
2213		return (LM_ERROR);		
2214		}		
2215		outfile = fopen(filename, "w");		
2216		if (outfile == NULL) {		
2217		lm_queue_message(ERROR_MSG, "can't open file: %s for write",		
2218		filename);		
2219		return (LM_ERROR);		
2220		}		
2221		lm_choose_coas(lm_system_selected->id);		
2222		{		
2223		lm_put_int(TEST_NETWORK_CMD);		
2224		lm_put_int(5); /* command */		
2225		if (!lm_end_put(lm_system_selected->id))		
2226		return (LM_ERROR);		
2227		if (lm_get_int() != TEST_NETWORK_ANS) {		
2228		lm_queue_message(ERROR_MSG, "internal error: wrong reply received from modeler: %s",		
2229		lm_system_selected->name);		
2230		return (LM_ERROR);		
2231		}		
2232		if ((ret = lm_dequeue_errors()) == LM_ERROR)		
2233		return (LM_ERROR);		
2234		{		
2235		total = lm_get_int();		
2236		for (i = 0; i < total; ++i) {		
2237		addr = lm_get_int();		
2238		count = lm_get_int();		
2239		(void) fprintf(outfile, "%08X %d\n", addr, count);		
2240		}		
2241		return (ret);		
2242		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/signal.c	DATE 5/23/89	PAGE # 1/66
LINE #	SOURCE TEXT			
1	/* SCCS ID: signal.c rev 3.1 4/24/89 at 07:36:14 */			
2	#include "signal.h"			
3	#include "common.h"			
4	#include "message.h"			
5	#include "lm_sfi.h"			
6	#ifdef SUNOS1_5			
7	#define LM_BADSIG (int (*)())-1			
8	#else			
9	#define LM_BADSIG (void (*)())-1			
10	#endif			
11	#ifdef APOLLO			
12	int lm_call_atexit = LM_TRUE;			
13	#endif			
14	#ifdef SIGSTOP			
15	int lm_handle_SIGSTOP = LM_TRUE;			
16	#endif			
17	#ifdef SIGINT			
18	int lm_handle_SIGINT = LM_TRUE;			
19	#endif			
20	#ifdef SIGQUIT			
21	int lm_handle_SIGQUIT = LM_TRUE;			
22	#endif			
23	#ifdef SIGILL			
24	int lm_handle_SIGILL = LM_TRUE;			
25	#endif			
26	#ifdef SIGTRAP			
27	int lm_handle_SIGTRAP = LM_TRUE;			
28	#endif			
29	#ifdef SIGIOT			
30	int lm_handle_SIGIOT = LM_TRUE;			
31	#endif			
32	#ifdef SIGENT			
33	int lm_handle_SIGENT = LM_TRUE;			
34	#endif			
35	#ifdef SIGFPE			
36	int lm_handle_SIGFPE = LM_TRUE;			
37	#endif			
38	#ifdef SIGBUS			
39	int lm_handle_SIGBUS = LM_TRUE;			
40	#endif			
41	#ifdef SIGSEGV			
42	int lm_handle_SIGSEGV = LM_TRUE;			
43	#endif			
44	#ifdef SIGSYS			
45	int lm_handle_SIGSYS = LM_TRUE;			
46	#endif			
47	#ifdef SIGPIPE			
48	int lm_handle_SIGPIPE = LM_TRUE;			
49	#endif			
50	#ifdef SIGTERM			
51	int lm_handle_SIGTERM = LM_TRUE;			
52	#endif			
53	#ifdef SIGXCPU			
54	int lm_handle_SIGXCPU = LM_TRUE;			
55	#endif			
56	#ifdef SIGXFSZ			
57	int lm_handle_SIGXFSZ = LM_TRUE;			
58	#endif			
59	#ifdef SIGLOST			
60	int lm_handle_SIGLOST = LM_TRUE;			
61	#endif			
62	#ifdef SIGUSR1			
63	int lm_handle_SIGUSR1 = LM_TRUE;			
64	#endif			
65	#ifdef SIGUSR2			
66	int lm_handle_SIGUSR2 = LM_TRUE;			
67	#endif			
68	#ifdef SIGABRT			
69	int lm_handle_SIGABRT = LM_TRUE;			
70	#endif			
71	#ifdef SIGAPOLLO			
72	int lm_handle_SIGAPOLLO = LM_TRUE;			
73	#endif			
74	extern u_char lm_debug[];			
75	#ifdef DEBUG			
76	#define DPRINTF(x) (void)printf x			
77	#else			
78	#define DPRINTF(x) /* do nothing */			
79	#endif			
80	lm_handle_signals()			
81	{			
82	int error_condition;			
83	#ifdef SUNOS1_5			
84	int (*previous_signal)();			
85	int lm_global_signal_handler();			
86	#else			
87	void (*previous_signal)();			
88	void lm_global_signal_handler();			
89	#endif			
90	static int been_here_before = LM_FALSE;			
91	char *sig_failed = "call to set signal handler for %s failed";			
92	char *sig_previous = "overriding previously set signal handler for %s";			
93	extern int lm_end_modeling_session();			
94	if (been_here_before == LM_TRUE)			
95	return(SUCCESS);			
96	been_here_before = LM_TRUE;			
97	#ifdef DEBUG			
98	if (lm_debug[6]) {			
99	DPRINTF(("lm_handle_signals(): entering signal handler setup\n"));			
100	}			
101	#endif			
102	error_condition = LM_FALSE;			
103	#ifdef APOLLO			
104	if (lm_call_atexit == LM_TRUE) {			
105	#ifdef SUN			
106	#ifdef DEBUG			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/signal.c	DATE 5/23/89	PAGE # 2/67
LINE #		SOURCE TEXT		
121		if (lm_debug(6)) {		
122		DPRINTF(("lm_handle_signals(): setting up on_exit()\n"));		
123		}		
124		#endif		
125		if (on_exit(lm_end_modeling_session, (char *)NULL) != 0) {		
126		error_condition = LM_TRUE;		
127		lm_queue_message(SYS_ERROR_MSG,		
128		"call to set exit handler 'on_exit' failed");		
129		}		
130		#else /* VMS or PCISC */		
131		#ifdef DEBUG		
132		if (lm_debug(6)) {		
133		DPRINTF(("lm_handle_signals(): setting up atexit()\n"));		
134		}		
135		#endif		
136		if (atexit(lm_end_modeling_session) != 0) {		
137		error_condition = LM_TRUE;		
138		lm_queue_message(SYS_ERROR_MSG,		
139		"call to set exit handler 'atexit' failed");		
140		}		
141		#endif /* SUN */		
142		#endif		
143		#ifdef DEBUG		
144		else {		
145		if (lm_debug(6)) {		
146		DPRINTF(("lm_handle_signals(): NOT setting up [at on_]exit()\n"));		
147		}		
148		#endif		
149		#endif /* DEBUG */		
150		#endif /* APOLLO */		
151		}		
152		#endif		
153		#ifdef SIGHUP		
154		if (lm_handle_SIGHUP == LM_TRUE) {		
155		#ifdef DEBUG		
156		if (lm_debug(6)) {		
157		DPRINTF(("lm_handle_signals(): setting up handler for SIGHUP()\n"));		
158		}		
159		#endif		
160		previous_signal = signal(SIGHUP, lm_global_signal_handler);		
161		if (previous_signal == LM_BADSIG) {		
162		error_condition = LM_TRUE;		
163		lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGHUP");		
164		}		
165		else if ((previous_signal != SIG_DFL) &&		
166		(previous_signal != SIG_IGN) &&		
167		(previous_signal != lm_global_signal_handler))		
168		{		
169		lm_queue_message(WARNING_MSG, sig_previous, "SIGHUP");		
170		}		
171		}		
172		#endif		
173		#ifdef DEBUG		
174		else {		
175		if (lm_debug(6)) {		
176		DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGHUP()\n"));		
177		}		
178		#endif		
179		#endif		
180		}		
181		#ifdef SIGINT		
182		if (lm_handle_SIGINT == LM_TRUE) {		
183		#ifdef DEBUG		
184		if (lm_debug(6)) {		
185		DPRINTF(("lm_handle_signals(): setting up handler for SIGINT()\n"));		
186		}		
187		#endif		
188		previous_signal = signal(SIGINT, lm_global_signal_handler);		
189		if (previous_signal == LM_BADSIG) {		
190		error_condition = LM_TRUE;		
191		lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGINT");		
192		}		
193		else if ((previous_signal != SIG_DFL) &&		
194		(previous_signal != SIG_IGN) &&		
195		(previous_signal != lm_global_signal_handler))		
196		{		
197		lm_queue_message(WARNING_MSG, sig_previous, "SIGINT");		
198		}		
199		}		
200		#endif		
201		#ifdef DEBUG		
202		else {		
203		if (lm_debug(6)) {		
204		DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGINT()\n"));		
205		}		
206		#endif		
207		#endif		
208		}		
209		#ifdef SIGQUIT		
210		if (lm_handle_SIGQUIT == LM_TRUE) {		
211		#ifdef DEBUG		
212		if (lm_debug(6)) {		
213		DPRINTF(("lm_handle_signals(): setting up handler for SIGQUIT()\n"));		
214		}		
215		#endif		
216		previous_signal = signal(SIGQUIT, lm_global_signal_handler);		
217		if (previous_signal == LM_BADSIG) {		
218		error_condition = LM_TRUE;		
219		lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGQUIT");		
220		}		
221		else if ((previous_signal != SIG_DFL) &&		
222		(previous_signal != SIG_IGN) &&		
223		(previous_signal != lm_global_signal_handler))		
224		{		
225		lm_queue_message(WARNING_MSG, sig_previous, "SIGQUIT");		
226		}		
227		}		
228		#endif		
229		#ifdef DEBUG		
230		else {		
231		if (lm_debug(6)) {		
232		DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGQUIT()\n"));		
233		}		
234		#endif		
235		#endif		
236		}		
237		#ifdef SIGILL		
238		if (lm_handle_SIGILL == LM_TRUE) {		
239		#ifdef DEBUG		
240		if (lm_debug(6)) {		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/signal.c	DATE 5/23/89	PAGE # 3/68
SOURCE TEXT				
LINE #				
241	DPRINTF("lm_handle_signals(): setting up handler for SIGILL()\n");			
242	}			
243	#endif			
244	previous_signal = signal(SIGILL, lm_global_signal_handler);			
245	if (previous_signal == LM_BADSIG) {			
246	error_condition = LM_TRUE;			
247	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGILL");			
248	}			
249	else if ((previous_signal != SIG_DFL) &&			
250	(previous_signal != SIG_IGN) &&			
251	(previous_signal != lm_global_signal_handler))			
252	{			
253	lm_queue_message(WARNING_MSG, sig_previous, "SIGILL");			
254	}			
255	}			
256	#ifdef DEBUG			
257	else {			
258	if (lm_debug[6]) {			
259	DPRINTF("lm_handle_signals(): NOT setting up handler for SIGILL()\n");			
260	}			
261	}			
262	#endif			
263	#endif			
264	#endif			
265	#ifdef SIGTRAP			
266	if (lm_handle_SIGTRAP == LM_TRUE) {			
267	#ifdef DEBUG			
268	if (lm_debug[6]) {			
269	DPRINTF("lm_handle_signals(): setting up handler for SIGTRAP()\n");			
270	}			
271	#endif			
272	previous_signal = signal(SIGTRAP, lm_global_signal_handler);			
273	if (previous_signal == LM_BADSIG) {			
274	error_condition = LM_TRUE;			
275	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGTRAP");			
276	}			
277	else if ((previous_signal != SIG_DFL) &&			
278	(previous_signal != SIG_IGN) &&			
279	(previous_signal != lm_global_signal_handler))			
280	{			
281	lm_queue_message(WARNING_MSG, sig_previous, "SIGTRAP");			
282	}			
283	}			
284	#ifdef DEBUG			
285	else {			
286	if (lm_debug[6]) {			
287	DPRINTF("lm_handle_signals(): NOT setting up handler for SIGTRAP()\n");			
288	}			
289	}			
290	#endif			
291	#endif			
292	#endif			
293	#ifdef SIGIOT			
294	if (lm_handle_SIGIOT == LM_TRUE) {			
295	#ifdef DEBUG			
296	if (lm_debug[6]) {			
297	DPRINTF("lm_handle_signals(): setting up handler for SIGIOT()\n");			
298	}			
299	#endif			
300	previous_signal = signal(SIGIOT, lm_global_signal_handler);			
301	if (previous_signal == LM_BADSIG) {			
302	error_condition = LM_TRUE;			
303	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGIOT");			
304	}			
305	else if ((previous_signal != SIG_DFL) &&			
306	(previous_signal != SIG_IGN) &&			
307	(previous_signal != lm_global_signal_handler))			
308	{			
309	lm_queue_message(WARNING_MSG, sig_previous, "SIGIOT");			
310	}			
311	}			
312	#ifdef DEBUG			
313	else {			
314	if (lm_debug[6]) {			
315	DPRINTF("lm_handle_signals(): NOT setting up handler for SIGIOT()\n");			
316	}			
317	}			
318	#endif			
319	#endif			
320	#endif			
321	#ifdef SIGINT			
322	if (lm_handle_SIGINT == LM_TRUE) {			
323	#ifdef DEBUG			
324	if (lm_debug[6]) {			
325	DPRINTF("lm_handle_signals(): setting up handler for SIGINT()\n");			
326	}			
327	#endif			
328	previous_signal = signal(SIGINT, lm_global_signal_handler);			
329	if (previous_signal == LM_BADSIG) {			
330	error_condition = LM_TRUE;			
331	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGINT");			
332	}			
333	else if ((previous_signal != SIG_DFL) &&			
334	(previous_signal != SIG_IGN) &&			
335	(previous_signal != lm_global_signal_handler))			
336	{			
337	lm_queue_message(WARNING_MSG, sig_previous, "SIGINT");			
338	}			
339	}			
340	#ifdef DEBUG			
341	else {			
342	if (lm_debug[6]) {			
343	DPRINTF("lm_handle_signals(): NOT setting up handler for SIGINT()\n");			
344	}			
345	}			
346	#endif			
347	#endif			
348	#endif			
349	#ifdef SIGFPE			
350	if (lm_handle_SIGFPE == LM_TRUE) {			
351	#ifdef DEBUG			
352	if (lm_debug[6]) {			
353	DPRINTF("lm_handle_signals(): setting up handler for SIGFPE()\n");			
354	}			
355	#endif			
356	previous_signal = signal(SIGFPE, lm_global_signal_handler);			
357	if (previous_signal == LM_BADSIG) {			
358	error_condition = LM_TRUE;			
359	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGFPE");			
360	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/signal.c	DATE 5/23/89	PAGE # 4/69
LINE #	SOURCE TEXT			
361	else if ((previous_signal != SIG_DFL) &&			
362	(previous_signal != SIG_IGN) &&			
363	(previous_signal != lm_global_signal_handler))			
364	{			
365	lm_queue_message(WARNING_MSG, sig_previous, "SIGFPE");			
366	}			
367	}			
368	#ifdef DEBUG			
369	else {			
370	if (lm_debug[6]) {			
371	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGFPE()\n"));			
372	}			
373	}			
374	#endif			
375	#endif			
376	#endif			
377	#ifdef SIGBUS			
378	if (lm_handle_SIGBUS == LM_TRUE) {			
379	#ifdef DEBUG			
380	if (lm_debug[6]) {			
381	DPRINTF(("lm_handle_signals(): setting up handler for SIGBUS()\n"));			
382	}			
383	#endif			
384	previous_signal = signal(SIGBUS, lm_global_signal_handler);			
385	if (previous_signal == LM_BADSIG) {			
386	error_condition = LM_TRUE;			
387	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGBUS");			
388	}			
389	else if ((previous_signal != SIG_DFL) &&			
390	(previous_signal != SIG_IGN) &&			
391	(previous_signal != lm_global_signal_handler))			
392	{			
393	lm_queue_message(WARNING_MSG, sig_previous, "SIGBUS");			
394	}			
395	}			
396	#ifdef DEBUG			
397	else {			
398	if (lm_debug[6]) {			
399	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGBUS()\n"));			
400	}			
401	}			
402	#endif			
403	#endif			
404	#endif			
405	#ifdef SIGSEGV			
406	if (lm_handle_SIGSEGV == LM_TRUE) {			
407	#ifdef DEBUG			
408	if (lm_debug[6]) {			
409	DPRINTF(("lm_handle_signals(): setting up handler for SIGSEGV()\n"));			
410	}			
411	#endif			
412	previous_signal = signal(SIGSEGV, lm_global_signal_handler);			
413	if (previous_signal == LM_BADSIG) {			
414	error_condition = LM_TRUE;			
415	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGSEGV");			
416	}			
417	else if ((previous_signal != SIG_DFL) &&			
418	(previous_signal != SIG_IGN) &&			
419	(previous_signal != lm_global_signal_handler))			
420	{			
421	lm_queue_message(WARNING_MSG, sig_previous, "SIGSEGV");			
422	}			
423	}			
424	#ifdef DEBUG			
425	else {			
426	if (lm_debug[6]) {			
427	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGSEGV()\n"));			
428	}			
429	}			
430	#endif			
431	#endif			
432	#endif			
433	#ifdef SIGSYS			
434	if (lm_handle_SIGSYS == LM_TRUE) {			
435	#ifdef DEBUG			
436	if (lm_debug[6]) {			
437	DPRINTF(("lm_handle_signals(): setting up handler for SIGSYS()\n"));			
438	}			
439	#endif			
440	previous_signal = signal(SIGSYS, lm_global_signal_handler);			
441	if (previous_signal == LM_BADSIG) {			
442	error_condition = LM_TRUE;			
443	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGSYS");			
444	}			
445	else if ((previous_signal != SIG_DFL) &&			
446	(previous_signal != SIG_IGN) &&			
447	(previous_signal != lm_global_signal_handler))			
448	{			
449	lm_queue_message(WARNING_MSG, sig_previous, "SIGSYS");			
450	}			
451	}			
452	#ifdef DEBUG			
453	else {			
454	if (lm_debug[6]) {			
455	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGSYS()\n"));			
456	}			
457	}			
458	#endif			
459	#endif			
460	#endif			
461	#ifdef SIGPIPE			
462	if (lm_handle_SIGPIPE == LM_TRUE) {			
463	#ifdef DEBUG			
464	if (lm_debug[6]) {			
465	DPRINTF(("lm_handle_signals(): setting up handler for SIGPIPE()\n"));			
466	}			
467	#endif			
468	previous_signal = signal(SIGPIPE, lm_global_signal_handler);			
469	if (previous_signal == LM_BADSIG) {			
470	error_condition = LM_TRUE;			
471	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGPIPE");			
472	}			
473	else if ((previous_signal != SIG_DFL) &&			
474	(previous_signal != SIG_IGN) &&			
475	(previous_signal != lm_global_signal_handler))			
476	{			
477	lm_queue_message(WARNING_MSG, sig_previous, "SIGPIPE");			
478	}			
479	}			
480	#ifdef DEBUG			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/signal.c

DATE 5/23/89
TIME 1:20:58 pm

PAGE #
5/70

LINE #	SOURCE TEXT
481	else {
482	if (lm_debug[6]) {
483	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGPIPE()\n"));
484	}
485	}
486	endif
487	endif
488	endif
489	endif
490	if (lm_handle_SIGTERM == LM_TRUE) {
491	endif
492	if (lm_debug[6]) {
493	DPRINTF(("lm_handle_signals(): setting up handler for SIGTERM()\n"));
494	}
495	endif
496	previous_signal = signal(SIGTERM, lm_global_signal_handler);
497	if (previous_signal == LM_BADSIG) {
498	error_condition = LM_TRUE;
499	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGTERM");
500	}
501	else if ((previous_signal != SIG_DFL) &&
502	(previous_signal != SIG_IGN) &&
503	(previous_signal != lm_global_signal_handler))
504	{
505	lm_queue_message(WARNING_MSG, sig_previous, "SIGTERM");
506	}
507	}
508	endif
509	else {
510	if (lm_debug[6]) {
511	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGTERM()\n"));
512	}
513	}
514	endif
515	endif
516	endif
517	endif
518	if (lm_handle_SIGICPU == LM_TRUE) {
519	endif
520	if (lm_debug[6]) {
521	DPRINTF(("lm_handle_signals(): setting up handler for SIGICPU()\n"));
522	}
523	endif
524	previous_signal = signal(SIGICPU, lm_global_signal_handler);
525	if (previous_signal == LM_BADSIG) {
526	error_condition = LM_TRUE;
527	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGICPU");
528	}
529	else if ((previous_signal != SIG_DFL) &&
530	(previous_signal != SIG_IGN) &&
531	(previous_signal != lm_global_signal_handler))
532	{
533	lm_queue_message(WARNING_MSG, sig_previous, "SIGICPU");
534	}
535	}
536	endif
537	else {
538	if (lm_debug[6]) {
539	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGICPU()\n"));
540	}
541	}
542	endif
543	endif
544	endif
545	endif
546	if (lm_handle_SIGFPE == LM_TRUE) {
547	endif
548	if (lm_debug[6]) {
549	DPRINTF(("lm_handle_signals(): setting up handler for SIGFPE()\n"));
550	}
551	endif
552	previous_signal = signal(SIGFPE, lm_global_signal_handler);
553	if (previous_signal == LM_BADSIG) {
554	error_condition = LM_TRUE;
555	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGFPE");
556	}
557	else if ((previous_signal != SIG_DFL) &&
558	(previous_signal != SIG_IGN) &&
559	(previous_signal != lm_global_signal_handler))
560	{
561	lm_queue_message(WARNING_MSG, sig_previous, "SIGFPE");
562	}
563	}
564	endif
565	else {
566	if (lm_debug[6]) {
567	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGFPE()\n"));
568	}
569	}
570	endif
571	endif
572	endif
573	endif
574	if (lm_handle_SIGLOST == LM_TRUE) {
575	endif
576	if (lm_debug[6]) {
577	DPRINTF(("lm_handle_signals(): setting up handler for SIGLOST()\n"));
578	}
579	endif
580	previous_signal = signal(SIGLOST, lm_global_signal_handler);
581	if (previous_signal == LM_BADSIG) {
582	error_condition = LM_TRUE;
583	lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGLOST");
584	}
585	else if ((previous_signal != SIG_DFL) &&
586	(previous_signal != SIG_IGN) &&
587	(previous_signal != lm_global_signal_handler))
588	{
589	lm_queue_message(WARNING_MSG, sig_previous, "SIGLOST");
590	}
591	}
592	endif
593	else {
594	if (lm_debug[6]) {
595	DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGLOST()\n"));
596	}
597	}
598	endif
599	endif
600	endif

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/signal.c

DATE 5/23/89
TIME 1:20:58 pm

PAGE #
6/71

```

LINE #          SOURCE TEXT
601 #ifdef SIGUSR1
602     if (lm_handle_SIGUSR1 == LM_TRUE) {
603 #ifdef DEBUG
604         if (lm_debug[6]) {
605             DPRINTF(("lm_handle_signals(): setting up handler for SIGUSR1()\n"));
606         }
607     #endif
608     previous_signal = signal(SIGUSR1, lm_global_signal_handler);
609     if (previous_signal == LM_BADSIG) {
610         error_condition = LM_TRUE;
611         lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGUSR1");
612     }
613     else if ( (previous_signal != SIG_DFL) &&
614              (previous_signal != SIG_IGN) &&
615              (previous_signal != lm_global_signal_handler) )
616     {
617         lm_queue_message(WARNING_MSG, sig_previous, "SIGUSR1");
618     }
619 #endif
620 #ifdef DEBUG
621     else {
622         if (lm_debug[6]) {
623             DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGUSR1()\n"));
624         }
625     }
626 #endif
627 #endif
628 #endif
629 #ifdef SIGUSR2
630     if (lm_handle_SIGUSR2 == LM_TRUE) {
631 #ifdef DEBUG
632         if (lm_debug[6]) {
633             DPRINTF(("lm_handle_signals(): setting up handler for SIGUSR2()\n"));
634         }
635     #endif
636     previous_signal = signal(SIGUSR2, lm_global_signal_handler);
637     if (previous_signal == LM_BADSIG) {
638         error_condition = LM_TRUE;
639         lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGUSR2");
640     }
641     else if ( (previous_signal != SIG_DFL) &&
642              (previous_signal != SIG_IGN) &&
643              (previous_signal != lm_global_signal_handler) )
644     {
645         lm_queue_message(WARNING_MSG, sig_previous, "SIGUSR2");
646     }
647 #endif
648 #ifdef DEBUG
649     else {
650         if (lm_debug[6]) {
651             DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGUSR2()\n"));
652         }
653     }
654 #endif
655 #endif
656 #endif
657 #ifdef SIGABRT
658     if (lm_handle_SIGABRT == LM_TRUE) {
659 #ifdef DEBUG
660         if (lm_debug[6]) {
661             DPRINTF(("lm_handle_signals(): setting up handler for SIGABRT()\n"));
662         }
663     #endif
664     previous_signal = signal(SIGABRT, lm_global_signal_handler);
665     if (previous_signal == LM_BADSIG) {
666         error_condition = LM_TRUE;
667         lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGABRT");
668     }
669     else if ( (previous_signal != SIG_DFL) &&
670              (previous_signal != SIG_IGN) &&
671              (previous_signal != lm_global_signal_handler) )
672     {
673         lm_queue_message(WARNING_MSG, sig_previous, "SIGABRT");
674     }
675 #endif
676 #ifdef DEBUG
677     else {
678         if (lm_debug[6]) {
679             DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGABRT()\n"));
680         }
681     }
682 #endif
683 #endif
684 #endif
685 #ifdef SIGAPOLLO
686     if (lm_handle_SIGAPOLLO == LM_TRUE) {
687 #ifdef DEBUG
688         if (lm_debug[6]) {
689             DPRINTF(("lm_handle_signals(): setting up handler for SIGAPOLLO()\n"));
690         }
691     #endif
692     previous_signal = signal(SIGAPOLLO, lm_global_signal_handler);
693     if (previous_signal == LM_BADSIG) {
694         error_condition = LM_TRUE;
695         lm_queue_message(SYS_ERROR_MSG, sig_failed, "SIGAPOLLO");
696     }
697     else if ( (previous_signal != SIG_DFL) &&
698              (previous_signal != SIG_IGN) &&
699              (previous_signal != lm_global_signal_handler) )
700     {
701         lm_queue_message(WARNING_MSG, sig_previous, "SIGAPOLLO");
702     }
703 #endif
704 #ifdef DEBUG
705     else {
706         if (lm_debug[6]) {
707             DPRINTF(("lm_handle_signals(): NOT setting up handler for SIGAPOLLO()\n"));
708         }
709     }
710 #endif
711 #endif
712 #endif
713     if (error_condition == LM_TRUE)
714         return(FAILURE);
715     else return(SUCCESS);
716 }
717
718 #ifdef SUNOS3_5
719 int

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/signal.c	DATE 5/23/89	PAGE # 7/72
LINE #	SOURCE TEXT			
721	/*else			
722	void			
723	endif			
724	lm_global_signal_handler(sig)			
725	int sig,			
726	{			
727	#ifdef DEBUG			
728	if (lm_debug(6)) {			
729	printf("lm_global_signal_handler(): called for signal '%d'\n", sig);			
730	}			
731	endif			
732	{void} lm_end_modeling_session();			
733	{void} signal(sig, SIG_DFL);			
734	{void} kill(getpid(), sig);			
735	}			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/utill.c

DATE 5/23/89
TIME 1:20:59 pm

PAGE #
1/73

LINE # SOURCE TEXT

```

1 // SCS_ID: utill.c rev 3.4, 5/9/89 at 16:26:37
2 #include <stdio.h>
3 #include <ctype.h>
4 #include "common.h"
5 #include "messages.h"
6 #include "network.h"
7 #include "lm_sfi.h"
8 #include "lrfi.h"
9 #include "protase.h"
10 #include "protcd.h"
11 #include "conv.h"
12
13 extern char *calloc();
14 extern char *lm_calloc();
15 extern char *lm_malloc();
16 extern char *lm_realloc();
17
18 #define MAX_INST_ID_TABLE_SIZE 128
19 #define MAX_INST_ID_TABLE_INCR 64
20
21 #define MAX_DEF_ID_TABLE_SIZE 128
22 #define MAX_DEF_ID_TABLE_INCR 64
23
24 #define MAX_DAB_PIN_ID_TABLE_SIZE 80
25 #define MAX_DAB_PIN_ID_TABLE_INCR 80
26
27 #define MAX_USER_PIN_ID_TABLE_SIZE 80
28 #define MAX_USER_PIN_ID_TABLE_INCR 40
29
30 #define PTR_SIZE 4
31
32 lm_init()
33 {
34     INSTANCE_INFO **temp_ptr,
35     u_short i,
36     extern char *getenv();
37
38 #ifdef DEBUG
39     char *temp,
40     long number;
41
42     // Initialize lm_debug flag //
43     for (i = 0; i < MAX_DEBUG; ++i)
44         lm_debug[i] = 0;
45
46     temp = (char *) getenv("HOSTDEBUG");
47     if (temp != NULL) {
48         i = 0;
49         // skip leading blank //
50         while (temp[i] == ' ')
51             i++;
52         while (temp[i]) {
53             if (sscanf(temp[i], "%d", &number)) {
54                 if ((number < 0) || (number > MAX_DEBUG)) {
55                     (void) printf("illegal lm_debug number: %d\n", number);
56                     exit(1);
57                 }
58                 (void) printf("Setting lm_debug number: %d\n", number);
59                 lm_debug[number] = 1;
60             }
61             else {
62                 (void) printf("invalid decimal number in : %s\n", temp[i]);
63                 exit(1);
64             }
65             while ((temp[i] != ' ') && (temp[i] != '\0'))
66                 i++;
67             while ((temp[i] == ' ') && (temp[i] != '\0'))
68                 i++;
69         }
70     }
71 }
72 #endif
73
74 #ifdef DEBUG
75     temp = (char *) getenv("NETWORK_TIMEOUT");
76     if (temp != NULL) {
77         if (sscanf(temp, "%d", &number)) {
78             DPRINTF(("set network timeout to: %d\n", number));
79             lm_network_timeout = number;
80         }
81     }
82 #endif
83
84 // Create the SYSTEM table //
85 lm_system_table = (SYSTEM_INFO **)
86     DREALLOC((unsigned) (MAX_SYSTEM_TABLE_SIZE * sizeof(SYSTEM_INFO *)));
87
88 if (lm_system_table == NULL) {
89     lm_queue_message(ERROR_MSG, "out of memory on host for system table");
90     return (FAILURE);
91 }
92 for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i)
93     lm_system_table[i] = NULL;
94
95 // Create the DEFINITION table //
96 lm_def_id_table = (DEFINITION_INFO **)
97     DREALLOC((unsigned) (MAX_DEF_ID_TABLE_SIZE * PTR_SIZE));
98 if (lm_def_id_table == NULL) {
99     lm_queue_message(ERROR_MSG, "out of memory on host for definition table");
100     return (FAILURE);
101 }
102 for (i = 0; i < MAX_DEF_ID_TABLE_SIZE; ++i)
103     lm_def_id_table[i] = NULL;
104 lm_def_id_table_size = MAX_DEF_ID_TABLE_SIZE;
105 lm_def_id_avail = 0;
106
107 // Create the INSTANCE table //
108 lm_inst_id_table = (INSTANCE_INFO **)
109     DREALLOC((unsigned) (MAX_INST_ID_TABLE_SIZE * PTR_SIZE));
110
111 if (lm_inst_id_table == NULL) {
112     lm_queue_message(ERROR_MSG, "out of memory on host for instance table");
113     return (FAILURE);
114 }
115 lm_free_instance_id = lm_inst_id_table;
116 temp_ptr = (INSTANCE_INFO **) &lm_inst_id_table[1];
117
118 for (i = 0; i < (MAX_INST_ID_TABLE_SIZE - 1); ++i) {
119     lm_inst_id_table[i] = (INSTANCE_INFO *) temp_ptr;
120 }

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM sfi/utill.c	DATE 5/23/89	PAGE # 2/74
LINE #		SOURCE TEXT		
121		++temp_ptr;		
122		{		
123		lm_inst_id_table[i] = NULL;		
124		lm_inst_id_table_size = MAX_INST_ID_TABLE_SIZE;		
125		/* Create the TEMPORARY Buffer for DAB Def storage */		
126		lm_temp_buffer = (char *) MALLOC((unsigned) (MAX_COMP_BUFFER_SIZE));		
127		if (lm_temp_buffer == NULL) {		
128		lm_queue_message(ERROR_MSG, "out of memory on host for temp buffer");		
129		return (FAILURE);		
130		}		
131		lm_buf_ptr = lm_temp_buffer + 4; /* the first inst location is for the		
132		length */		
133		lm_max_buf_addr = lm_temp_buffer + MAX_COMP_BUFFER_SIZE;		
134		/* initialize logic map */		
135		for (i = 0; i < MAX_LOGIC_STATE; ++i)		
136		lm_user_to_lm_logic_level_map[i].set = 0;		
137		if (lm_open_modelers_list_file() == FAILURE)		
138		return (FAILURE);		
139		return (SUCCESS);		
140		}		
141		lm_uninit()		
142		{		
143		register int i, fd;		
144		register SYSTEM_INFO *system;		
145		{		
146		(void) release_all_instances();		
147		DFREE((char *) lm_inst_id_table);		
148		for (i = 0; i < MAX_DEF_ID_TABLE_SIZE; ++i) {		
149		if (lm_def_id_table[i] != NULL) {		
150		xls_definition_info((u_short) i);		
151		}		
152		lm_last_definition_id_specified = -1;		
153		lm_definition_selected = NULL;		
154		DFREE((char *) lm_def_id_table);		
155		for (i = 0; i < MAX_SYSTEM_TABLE_SIZE; ++i) {		
156		if ((system = lm_system_table[i]) != NULL) {		
157		if (system->name != NULL) {		
158		DFREE(system->name);		
159		}		
160		switch (system->state) {		
161		case USED:		
162		case USED_FOR_RD_WR:		
163		if ((fd = system->fd) != -1) {		
164		lm_close_modeler(fd);		
165		}		
166		}		
167		DFREE((char *) system);		
168		}		
169		DFREE((char *) lm_system_table);		
170		if (lm_temp_buffer != NULL)		
171		DFREE((char *) lm_temp_buffer);		
172		}		
173		static		
174		release_all_instances()		
175		{		
176		INSTANCE_INFO *instance;		
177		FILE_INFO *file_ptr;		
178		u_short instance_id;		
179		{		
180		clear_errors();		
181		lm_last_instance_id_specified = -1;		
182		lm_instance_selected = NULL;		
183		for (instance_id = 0; instance_id < lm_inst_id_table_size; ++instance_id) {		
184		instance = lm_inst_id_table[instance_id];		
185		if (BOGUS_INSTANCE(instance))		
186		continue;		
187		if (lm_search_key_id(lm_timing_file_ptr,		
188		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
189		lm_close_and_delete_file_key_id(lm_timing_file_ptr,		
190		(u_short) instance->inst_id);		
191		instance->tm_file = NULL;		
192		if (lm_search_key_id(lm_vector_file_ptr,		
193		(u_short) instance->inst_id, &file_ptr) == SUCCESS) {		
194		lm_close_and_delete_file_key_id(lm_vector_file_ptr,		
195		(u_short) instance->inst_id);		
196		instance->vector_file = NULL;		
197		}		
198		if (xls_instance_info(lm_def_id_table[instance->def_id], instance);		
199		return (LM_SUCCESS);		
200		}		
201		static		
202		extend_def_id_table()		
203		{		
204		char *ptr;		
205		u_short i;		
206		u_short new_table_size;		
207		{		
208		#ifdef DEBUG		
209		if (lm_debug[1]) {		
210		PRINTF("inside extend_def_id_table()\n");		
211		}		
212		#endif		
213		new_table_size = lm_def_id_table_size + MAX_DEF_ID_TABLE_INCR;		
214		ptr = (char *) MALLOC((char *) lm_def_id_table,		
215		(unsigned) (new_table_size * PTR_SIZE));		
216		if (ptr == NULL) {		
217		lm_queue_message(ERROR_MSG, "out of memory on host for definition table");		
218		return (FAILURE);		
219		}		
220		}		

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

sfi/util1.c

DATE

5/23/89

PAGE #

TIME

1:20:59 pm

3/75

```

241  lm_def_id_table = (DEFINITION_INFO **) ptr;
242
243  for (i = lm_def_id_table_size; i < new_table_size; ++i)
244    lm_def_id_table[i] = NULL;
245
246  lm_def_id_table_size = new_table_size;
247  return (SUCCESS);
248 }
249
250 lm_extend_inst_id_table()
251 {
252     INSTANCE_INFO **temp_ptr;
253     char *ptr;
254     u_short i;
255     u_short new_table_size;
256
257 #ifdef DEBUG
258     if (lm_debug[1]) {
259         fprintf(stderr, "inside lm_extend_inst_id_table\n");
260     }
261 #endif
262
263     new_table_size = lm_inst_id_table_size + MAX_INST_ID_TABLE_INCR;
264     ptr = (char *) DREALLOC((char *) lm_inst_id_table,
265                             (unsigned) (new_table_size * PTR_SIZE));
266     if (ptr == NULL) {
267         lm_queue_message(ERROR_MSG, "out of memory on host for instance table");
268         return (FAILURE);
269     }
270     lm_inst_id_table = (INSTANCE_INFO **) ptr;
271
272     lm_free_instance_id = (INSTANCE_INFO *)
273         &lm_inst_id_table[lm_inst_id_table_size];
274
275     temp_ptr = lm_free_instance_id + 1;
276     for (i = lm_inst_id_table_size; i < (new_table_size - 1); ++i) {
277         lm_inst_id_table[i] = (INSTANCE_INFO *) temp_ptr;
278         temp_ptr++;
279     }
280     lm_inst_id_table[i] = NULL;
281
282     lm_inst_id_table_size = new_table_size;
283     return (SUCCESS);
284 }
285
286 lm_extend_dab_pid_table(definition, pin_number)
287     DEFINITION_INFO *definition;
288     u_long pin_number;
289 {
290     DAB_PID *temp_ptr;
291     char *ptr;
292     u_long new_size;
293     u_short i;
294
295 #ifdef DEBUG
296     if (lm_debug[1]) {
297         fprintf(stderr, "inside lm_extend_dab_pid_table\n");
298     }
299 #endif
300
301     new_size = definition->dab_pid_table_size + MAX_DAB_PIN_ID_TABLE_INCR;
302     while (pin_number >= new_size) {
303         new_size += MAX_DAB_PIN_ID_TABLE_INCR;
304     }
305     ptr = (char *) DREALLOC((char *) definition->dab_pid_table,
306                             (unsigned) (new_size * sizeof(DAB_PID)));
307     if (ptr == NULL) {
308         lm_queue_message(ERROR_MSG, "out of memory on host for Device Adapter pin_id table");
309         return (FAILURE);
310     }
311     definition->dab_pid_table = (DAB_PID *) ptr;
312
313     temp_ptr = (definition->dab_pid_table + definition->dab_pid_table_size);
314     for (i = definition->dab_pid_table_size; i < new_size; ++i) {
315         temp_ptr->name = NULL;
316         temp_ptr->key = (INVALID_KEY - LM_INPUT) << PIN_TYPE_BIT_OFFSET;
317         temp_ptr->pin_type = NONE;
318         temp_ptr->pin_number = INVALID_PIN_NUMBER;
319         temp_ptr++;
320     }
321     definition->dab_pid_table_size = new_size;
322     return (SUCCESS);
323 }
324
325 lm_extend_tbuf()
326 {
327     char *ptr;
328     u_short temp;
329     u_short new_size;
330
331 #ifdef DEBUG
332     if (lm_debug[1]) {
333         fprintf(stderr, "inside lm_extend_tbuf\n");
334     }
335 #endif
336
337     temp = (u_short) (lm_tbuf_ptr - lm_temp_buffer);
338     new_size = lm_max_tbuf_addr - lm_temp_buffer + MAX_COMM_BUFFER_INCR;
339     ptr = (char *) DREALLOC((char *) lm_temp_buffer, (unsigned) new_size);
340     if (ptr == NULL) {
341         lm_queue_message(ERROR_MSG, "out of memory on host for temp buffer");
342         return (FAILURE);
343     }
344     lm_temp_buffer = ptr;
345     lm_tbuf_ptr = lm_temp_buffer + temp;
346     lm_max_tbuf_addr = lm_temp_buffer + new_size;
347     return (SUCCESS);
348 }
349
350 static
351 new_system_info(sys_ptr)
352     SYSTEM_INFO **sys_ptr;
353 {
354     SYSTEM_INFO *temp;
355
356     temp = (SYSTEM_INFO *) DREALLOC((unsigned) 1,
357                                     (unsigned) (sizeof(SYSTEM_INFO)));
358 }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
sfi/utill.c

DATE 5/23/89
TIME 1:20:59 pm

PAGE #
4/76

```

LINE # SOURCE TEXT
361 if (temp == NULL) {
362     lm_queue_message(ERROR_MSG, "out of memory as host for system_info",
363         return (FAILURE);
364     }
365     temp = NULL;
366     temp->id = -1;
367     temp->state = NOT_OPENED;
368     *sys_ptr = temp;
369     return (SUCCESS);
370 }
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_boot.c

DATE

5/23/89

PAGE #

TIME

1:20:46 pm

1/1

LINE #	SOURCE TEXT
1	/* SCCS ID: net_boot.c rev 3.1, 4/24/89 at 08:02:31 */
2	#include <stdio.h>
3	#include "common.h"
4	#include "message.h"
5	#include "network.h"
6	#include "lm_rd_wr.h"
7	#include "lm_err.h"
8	
9	
10	u_short process_file();
11	void aecii_to_hex_string();
12	u_short a_rec_rcvd();
13	
14	#define MAX_SREC_SIZE 35
15	#define SREC_BUF_SIZE 255
16	
17	static char start_of_file;
18	static char srec_data(SREC_BUF_SIZE);
19	static char buffer(MAX_RD_WR_BUFFER_SIZE);
20	
21	static u_long count;
22	static u_long address;
23	static u_long start_execution;
24	
25	
26	lm_boot(modeler, number_of_files, file_list, jump, jump_address)
27	{
28	char *modeler;
29	u_short number_of_files;
30	char *file_list[];
31	u_short jump;
32	u_long jump_address;
33	u_long i, status;
34	/* initialize globals to allow multiple boots per session */
35	count = 0;
36	address = 0;
37	start_execution = 0;
38	/* set up data structures for lm_writes */
39	if (lm_open(modeler) == LM_ERROR) return LM_ERROR;
40	/* for each file process it
41	for(i = 0; i < number_of_files; i++)
42	{
43	start_of_file = TRUE;
44	(void) printf("Transferring %s\n", file_list[i]);
45	if(process_file(modeler, file_list[i]) == FAILURE)
46	{
47	lm_close_connection_after_boot(modeler);
48	return(LM_ERROR);
49	}
50	if(jump != FALSE)
51	{
52	start_execution = jump_address;
53	if(lm_write(LM_CPURAM_MEMORY, (u_long)0, start_execution, (u_long)0, (char *)0, &status) == FAILURE)
54	{
55	lm_queue_message(ERROR_MSG, "Unable to set Program Counter on Modeler.");
56	lm_close_connection_after_boot(modeler);
57	return(LM_ERROR);
58	}
59	lm_close_connection_after_boot(modeler);
60	return(LM_SUCCESS);
61	}
62	}
63	static u_short process_file(modeler, file_name)
64	{
65	char *modeler;
66	char *file_name;
67	u_long status;
68	int ch;
69	char *ptr_srec = srec_data;
70	FILE *fp, *fopen();
71	if((fp = fopen(file_name, "r")) == NULL)
72	{
73	lm_queue_message(SYS_ERROR_MSG, file_name);
74	return(FAILURE);
75	}
76	/* read the file and process data
77	/*
78	if (process_library(modeler, fp) == LM_SUCCESS) {
79	(void) fclose(fp);
80	return LM_SUCCESS;
81	}
82	while ((ch=getc(fp)) != EOF) {
83	if((char)ch == 'S')
84	{
85	if(ptr_srec != srec_data)
86	{
87	if(a_rec_rcvd() == FAILURE)
88	{
89	(void) fclose(fp);
90	return(FAILURE);
91	}
92	ptr_srec = srec_data;
93	}
94	/* we better clear what we have collected
95	/*
96	if((MAX_RD_WR_BUFFER_SIZE - count) < MAX_SREC_SIZE)
97	{
98	if(lm_write(LM_CPURAM_MEMORY, (u_long) 0, (u_long) address, (u_long) count, buffer, &status) != LM_SUCCESS)
99	{
100	(void) fclose(fp);
101	return(FAILURE);
102	}
103	count = 0;
104	}
105	}
106	ptr_srec++ = (char)ch;
107	if(ptr_srec > (srec_data+SREC_BUF_SIZE))
108	{
109	return(FAILURE);
110	}
111	}
112	return(FAILURE);
113	
114	
115	
116	
117	
118	
119	
120	

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM networkh/net_boot.c	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">DATE</td> <td style="width: 50%;">5/23/89</td> </tr> <tr> <td>TIME</td> <td>1:20:46 pm</td> </tr> </table>	DATE	5/23/89	TIME	1:20:46 pm
DATE	5/23/89					
TIME	1:20:46 pm					

PAGE #
 2/2

LINE #	SOURCE TEXT
121	1
122	1
123	if(ptr_arcc != arcc_data)
124	{
125	if(u_rec_rvrd() == FAILURE)
126	{
127	(void) fclose(fp);
128	return(FAILURE);
129	}
130	if(count)
131	{
132	if(!u_write(LM_CPU_RAM_MEMORY, (u_long) 0, (u_long) address, (u_long) count, buffer, &status) != LM_SUCCESS)
133	{
134	(void) fclose(fp);
135	return(FAILURE);
136	}
137	count = 0;
138	}
139	(void) fclose(fp);
140	return (SUCCESS);
141	}
142	1
143	/*
144	** we have a complete S record
145	** place it in memory.
146	*/
147	static u_short
148	s_rec_rvrd()
149	{
150	register unsigned char checksum;
151	register u_short i;
152	register u_long addr;
153	/*
154	** not interested in the record
155	*/
156	if(arcc_data[1] > '3' arcc_data[1] < '1') return(SUCCESS);
157	/*
158	** get it from ascii to hex
159	** note: 'XX' stays ascii, where X=1,2, or 3.
160	*/
161	ascii_to_hex_string(arcc_data);
162	/*
163	** place in memory, do a sanity check
164	*/
165	checksum = 0;
166	for(i = 0; i <= arcc_data[2]; i++)
167	checksum += arcc_data[i + 2];
168	if(checksum != 0xff)
169	{
170	in_queue_message(ERROR_MSG, "Bad Checksum for S record");
171	return(FAILURE);
172	}
173	/*
174	** set up address pointer
175	*/
176	i = 2;
177	if(arcc_data[1] == '1')
178	{
179	/*addr = (u_long) *((unsigned short *) (arcc_data + 3));*/
180	addr = (u_long) (((u_short) ((u_char) arcc_data[3])) << 8) ((unsigned short) ((u_char) arcc_data[4])));
181	}
182	else
183	{
184	/*addr = (u_long) *((unsigned long *) (arcc_data + 3));*/
185	addr = (u_long) (((u_long) ((u_char) arcc_data[3])) << 24)
186	(((u_long) ((u_char) arcc_data[4])) << 16)
187	(((u_long) ((u_char) arcc_data[5])) << 8)
188	(((u_long) ((u_char) arcc_data[6]))));
189	i = 4;
190	}
191	if(arcc_data[1] == '2')
192	{
193	addr = (addr >> 8);
194	/*
195	addr = (u_long) (((u_long) (arcc_data[3])) << 16)
196	(((u_long) (arcc_data[4])) << 8)
197	(((u_long) (arcc_data[5]))));
198	*/
199	i = 3;
200	}
201	if(count == 0)
202	address = addr;
203	/*
204	** load the data
205	*/
206	arcc_data[2];
207	if(start_of_file == TRUE)
208	{
209	if(start_offset < address)
210	start_offset = address;
211	start_of_file = FALSE;
212	}
213	/*
214	** save the data past the LM header
215	*/
216	for(i < arcc_data[2]; i++)
217	buffer[count++] = arcc_data[3 + i];
218	return (SUCCESS);
219	}
220	1
221	/*
222	** Turn a S record ascii string to hex
223	*/
224	#define ascii_to_hex_byte(c) (((c) >= '0') && ((c) <= '9')) ? \
225	(char) ((c) - '0') : (char) ((c) + 10 - 'A')
226	static void
227	ascii_to_hex_string(arcc_data)
228	{
229	register char *arcc_data;
230	register char byte_count, cnt;
231	register char bh, bl;
232	register char *from = (arcc_data + 2);
233	arcc_data += 2;
234	bh = *from++;
235	bl = *from++;
236	byte_count = (char)(ascii_to_hex_byte(bh) << 4) (ascii_to_hex_byte(bl));

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_boot.c

\$

DATE

5/23/89

PAGE #

TIME

1:20:46 pm

3/3

```

LINE #          SOURCE TEXT
241      *area_data++ = byte_count;
242      for (cat = 0, cat < byte_count, ++cat) {
243          bh = *from++;
244          bl = *from++;
245          *area_data++
246              = (ascii_to_hex_byte(bh)<<4)|(ascii_to_hex_byte(bl));
247      }
248  }
249
250  /*
251  **--labinary file is very similar to an .out file on a sun. (in fact
252  ** it is 100% compatible).
253  **
254  ** This is the .sun .out header definition and useful macros.
255  **/
256
257  struct exec {
258
259      unsigned short  a_machtype;
260      unsigned short  a_magic;
261      unsigned long   a_text;
262      unsigned long   a_data;
263      unsigned long   a_bss;
264      unsigned long   a_syms;
265      unsigned long   a_entry;
266      unsigned long   a_trsize;
267      unsigned long   a_drsize;
268  };
269
270  #define OMAGIC 0407          /* old impure format */
271  #define NMAGIC 0410         /* read-only text */
272  #define ZMAGIC 0413         /* demand load format */
273
274  #define N_BADMAG(x) \
275      ((x).a_magic != OMAGIC && (x).a_magic != NMAGIC && (x).a_magic != ZMAGIC)
276
277  static
278  process_labinary(modeler, fp)
279  char *modeler;
280  FILE *fp;
281  {
282      struct exec header;
283      register u_long size, addr;
284      char buffer[MAX_RD_WR_BUFFER_SIZE];
285      u_long status;
286
287      if (fread((char *)&header, 1, sizeof(header), fp) != sizeof(header)) {
288          goto abort_binary;
289      }
290      if (N_BADMAG(header) != 0) {
291          goto abort_binary;
292      }
293
294      addr = header.a_entry;
295      size = header.a_text + header.a_data;
296      if (start_execution < addr) start_execution = addr;
297
298      while (size > 0) {
299          count = (size < MAX_RD_WR_BUFFER_SIZE)?
300              size: MAX_RD_WR_BUFFER_SIZE;
301
302          if ((count = fread(buffer, 1, (int)count, fp)) <= 0) {
303              lm_queue_message(ERROR_MSG, "Corrupted labinary file.");
304              goto abort_binary;
305          }
306          if (lm_write(LM_CPU_RAM_MEMORY, (u_long) 0,
307              (u_long)addr, (u_long)count,
308              buffer, &status) != LM_SUCCESS) {
309              lm_queue_message(ERROR_MSG, "Write to modeler failed.");
310              goto abort_binary;
311          }
312          size -= count;
313          addr += count;
314      }
315
316      return LM_SUCCESS;
317
318  abort_binary:
319      rewind(fp);
320      return FAILURE;
321  }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_clnt.c

DATE

5/23/89

PAGE #

TIME

1:20:47 pm

1/4

```

1  /* SCCS ID: net_clnt.c rev 3.3, 5/9/89 at 17:31:34 */
2  #include <stdio.h>
3  #include <errno.h>
4  #include <sys/types.h>
5  #include <sys/time.h>
6  #include <netinet/in.h>
7  #include <netdb.h>
8  #include "common.h"
9  #include "message.h"
10 #include "network.h"
11 #include "in_err.h"
12 #include "mod_err.h"
13 #include "svaram.h"
14
15 /*
16  * The following is a list of all the global variables that
17  * exist in the client network code. The higher level user
18  * can feel free to change these as he sees fit, but beware
19  * their usage is non-trivial.
20  */
21
22 /*
23  * These track the total number of message retries for the entire
24  * life of this routine (in the calling process), the total number
25  * of packets that have been sent, the total number of mismatched
26  * packet sequence numbers, and the total number of timeouts.
27  */
28 u_long in_total_number_of_msg_retries = 0;
29 u_long in_total_number_of_packets_sent = 0;
30 u_long in_total_number_of_mismatches = 0;
31 u_long in_total_number_of_timeouts = 0;
32
33 /*
34  * The following three variables control the dropping, duplication,
35  * and failure to re-send packets on timeout. Dropping and duplicating
36  * packets is only used during testing. No re-send on timeout is used
37  * only for very special communication during modular testing.
38  */
39 u_long in_drop_packet = FALSE;
40 u_long in_duplicate_packet = FALSE;
41 u_long in_re-send-on-timeout = TRUE;
42
43 /*
44  * The maximum number of attempts that will be made to send a request
45  * to the modular. After a time limit has expired (defined below)
46  * the host will re-send the current request to the modular if there
47  * has been less than or equal to in_max_timeout_retry_attempts to do so.
48  */
49 u_long in_max_timeout_retry_attempts = 3;
50
51 /*
52  * Issue a warning message at the following interval (frequency) when
53  * there have been that many timeouts or mismatched packets.
54  */
55 u_long in_network_warning_freq = 100;
56
57 /*
58  * The in_network_timeout time should be set to the maximum value that
59  * we ever expect to take to evaluate a given device. With 2M patterns
60  * at a pattern depth of 15000, that would be 14.0 seconds. There could be
61  * a maximum of 4 users (one per line) eating up that kind of memory;
62  * therefore, the total, longest wait time is 56.0 seconds. But wait,
63  * the product evaluation allows for 100 patterns per line with associated
64  * time of 47.8 seconds. This is obviously too long to wait, so go
65  * with a conservative 15 seconds along with the simple adaptive algorithm
66  * explained below.
67  *
68  * (Note that it takes 170 days (200150000) and 30 years (150000000)
69  * to evaluate the indicated pattern depths with an infinitely fast simulator.
70  * Bottom line, I like 15 seconds.)
71  *
72  * The time between request retries increases as the number of attempts
73  * to get a response from the modular increases (up to in_max_timeout_retry_attempts).
74  * Such that the total time before giving up is:
75  *
76  * summation from i equals 1 to in_max_timeout_retry_attempts
77  * of (i * in_network_timeout);
78  *
79  * 3 retries at 15 seconds per yields a total of 90 seconds,
80  * which is more than enough for the current generation patterns memory. With
81  * the above situation described above, gives that it is impossible to build
82  * patterns to the required depths at the slower frequencies.
83  *
84  * 10/10/88 - Elio - Well, it turns out that everybody whines and complains
85  * that they have to wait too long for a modular to timeout.
86  * So, we've decided to go with 10 seconds, for a total timeout
87  * of 60 seconds.
88  */
89 u_long in_network_timeout = 20000; /* 20 seconds */
90
91 extern u_short send_single_message_get_ack();
92 extern u_short send_multiple_message_get_ack();
93 extern u_short receive_multiple_message();
94 extern u_short receive_packet_with_timeout();
95 extern u_short receive_packet();
96 extern u_short send_packet();
97 extern u_short obtain_modulars_address();
98 extern u_short process_timeout();
99 extern u_short process_mismatch();
100
101 u_short
102 in_send_request_get_reply(conn)
103 register CONNECTION *conn;
104 {
105     register u_short error_condition;
106
107     /*
108      * Set the total number of bytes in the outgoing message buffer.
109      * It is absolutely critical that this be done here, before any
110      * further buffer processing is done as the latter processing
111      * changes the value of conn->outgoing_buffer_pointer.
112      */
113     BYTE_COUNT(conn->outgoing_buffer_base) = (u_long)conn->outgoing_buffer_pointer - (u_long)conn->outgoing_buffer_base;
114 }

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
networkh/net_clnt.c

DATE 5/23/89

PAGE #

TIME 1:20:47 pm

2/5

```

LINE # SOURCE TEXT
121 /* Initialize where send[sm]() places the acknowledge.
122 */
123 conn->incoming_buffer_pointer = conn->incoming_buffer_base,
124
125 /*
126 * Initialize where send[sm]() takes the request from
127 */
128 conn->outgoing_buffer_pointer = conn->outgoing_buffer_base,
129
130 error_condition = FALSE,
131 if (SITE_COUNT(conn->outgoing_buffer_base) <= MAX_CLIENT_PACKET) {
132     /* The request will fit into a single network packet. Set the
133     /* indication that this is the only packet for this request.
134     conn->header.partial_packet_count = 0,
135
136     /* Send the request as a message and wait for the acknowledge.
137     /* This could be a full or partial reply. It's checked below.
138     if (send_single_message_get_ack(conn) != SUCCESS) {
139         error_condition = TRUE,
140     }
141 }
142 else { /* The request will NOT fit into a single network packet.
143     /* Set the indication that this is a partial packet.
144     conn->header.partial_packet_count = SITE_COUNT(conn->outgoing_buffer_base),
145
146     /* Send the request as a multi-message and wait for the acknowledge.
147     /* This could be a full or partial reply. It's checked below.
148     if (send_multiple_message_get_ack(conn) != SUCCESS) {
149         error_condition = TRUE,
150     }
151 }
152
153 /*
154 * Overlay a packet header struct on the reply to check for a partial reply.
155 */
156 if (error_condition == FALSE) {
157     if (PARTIAL_PACKET_COUNT(conn->incoming_buffer_base) != 0) {
158         /* Received only a partial reply.
159         if (receive_multiple_message(conn) != SUCCESS) {
160             error_condition = TRUE,
161         }
162     }
163     /*
164     * Else, we have received the whole reply and no further
165     * processing need be done.
166     */
167 }
168
169 /*
170 * Properly set up the incoming and outgoing buffers
171 * for the upper software layers.
172 */
173 IN_RESET_INCOMING(conn),
174 IN_RESET_OUTGOING(conn),
175 IN_SET_END_INCOMING(conn),
176
177 if (error_condition == FALSE)
178     return(SUCCESS),
179 else return(FAILURE),
180 }
181
182 static u_short
183 send_single_message_get_ack(conn)
184 register CONNECTION *conn,
185 {
186     register
187         u_short rts,
188         char state,
189         u_long network_timeout, /* can't be made storage class register */
190         u_short number_of_message_attempts, /* can't be made storage class register */
191         u_long sm_sending_packet = -1,
192
193     static
194     if (sm_sending_packet != -1) {
195         reset_network_connection(conn),
196
197         PROCESS_ID(conn->outgoing_buffer_pointer) = conn->header.process_id,
198         SEQUENCE_NUMBER(conn->outgoing_buffer_pointer) = 0,
199         PARTIAL_PACKET_COUNT(conn->outgoing_buffer_pointer) = 0,
200         SITE_COUNT(conn->outgoing_buffer_pointer) = 16,
201
202         send_packet(conn, conn->outgoing_buffer_pointer),
203         (void) receive_packet_with_timeout(conn, 1000),
204         in_queue_message (BROADCAST_MSG, "Network connection lost due to user interrupt"),
205         sm_sending_packet = -1,
206         return(FAILURE),
207     }
208
209     sm_sending_packet = conn->header.sequence_number,
210
211     /*
212     * Increment the sequence number before sending a new message.
213     */
214     conn->header.sequence_number++,
215
216     /*
217     * Set the appropriate header information in the message.
218     */
219     PROCESS_ID(conn->outgoing_buffer_pointer) = conn->header.process_id,
220     SEQUENCE_NUMBER(conn->outgoing_buffer_pointer) = conn->header.sequence_number,
221     PARTIAL_PACKET_COUNT(conn->outgoing_buffer_pointer) = conn->header.partial_packet_count,
222
223     /*
224     * Reset the count of how many times we have tried to send the message.
225     */
226     number_of_message_attempts = 0,
227
228     /*
229     * Reset the length of time we decide to wait for an acknowledge.
230     */
231     network_timeout = ls_network_timeout,
232
233     /*
234     * Send the message to the address in conn.
235     */
236     try_again:
237     if (ls_drop_packet == FALSE) {
238         if (send_packet(conn, conn->outgoing_buffer_pointer) != SUCCESS) {
239             sm_sending_packet = -1,
240             return(FAILURE),
241         }
242     }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_clnt.c

DATE

5/23/89

PAGE #

TIME

1:20:47 pm

3/6

```

LINE # SOURCE TEXT
241 }
242 /* Next time through the loop, let's send the packet */
243 in_drop_packet = FALSE;
244
245 if (in_duplicate_packet == TRUE) {
246     if (send_packet(conn, conn->outgoing_buffer_pointer) != SUCCESS) {
247         in_sending_packet = -1;
248         return(FAILURE);
249     }
250 }
251
252 switch (receive_packet_with_timeout(conn, 1000)) {
253 case SUCCESS:
254     goto process_received_packet;
255 case NOT_SUCCESS_OR_FAILURE:
256     /* Is the modalar alive? */
257     if (in_is_modalar_alive(conn->name, &state) != IN_SUCCESS) {
258         in_sending_packet = -1;
259         return(FAILURE);
260     }
261     break;
262 default:
263     in_sending_packet = -1;
264     return(FAILURE);
265 }
266
267 try_try_again_without_send:
268 rts = receive_packet_with_timeout(conn, network_timeout);
269 if (rts == SUCCESS) {
270     /* Fall through to below to process the received packet */
271 }
272 else if (rts == NOT_SUCCESS_OR_FAILURE) {
273     /* We have timed out and must resend the request */
274     if (process_timeout(&number_of_message_attempts,
275         network_timeout) != SUCCESS) {
276         in_sending_packet = -1;
277         return(FAILURE);
278     }
279 }
280
281 if (in_resend_on_timeout == TRUE)
282     goto try_try_again;
283 else {
284     in_sending_packet = -1;
285     return(SUCCESS);
286 }
287
288 }
289 else {
290     in_sending_packet = -1;
291     return(FAILURE);
292 }
293
294 process_received_packet:
295 /*
296  * Verify that the acknowledge is for the correct message.
297  */
298 if (conn->header.sequence_number == SEQUENCE_NUMBER(conn->incoming_buffer_pointer)) {
299     /* The message and acknowledge sequence numbers match - great! */
300     in_sending_packet = -1;
301     return(SUCCESS);
302 }
303
304 else if (SEQUENCE_NUMBER(conn->incoming_buffer_pointer) == conn->header.sequence_number - 1) {
305     /*
306      * The message and acknowledge sequence numbers don't match - ouch!
307      * Must be a duplicated previous reply. Go back to the point of packet
308      * receipt and receive the real (new) reply. Note that we do not
309      * resend the request as we assume the modalar has already received it
310      * and replied in the (to be) received reply. If the modalar never
311      * received the request, we will time out and resend as usual.
312      */
313     /* Process mismatched sequence numbers exactly as if a time-out */
314     /* had occurred. This checks for too many attempts at a correct
315      * acknowledge and may possibly give up if things don't straighten out.
316      */
317     if (process_mismatch() != SUCCESS) {
318         in_sending_packet = -1;
319         return(FAILURE);
320     }
321     goto try_try_again_without_send;
322 }
323
324 else {
325     /* The whole world must be messed up - giving up! */
326     in_queue_message(ERROR_MSG, "mismatched message transaction sequence numbers from the modalar(%d, %d)",
327         SEQUENCE_NUMBER(conn->incoming_buffer_pointer),
328         conn->header.sequence_number);
329 }
330
331 in_sending_packet = -1;
332 return(FAILURE);
333
334 static u_short
335 send_multiple_message_get_ack(conn)
336 register CONNECTION *conn;
337 {
338     register u_long packet_byte_count;
339     register u_long message_bytes_left_to_send;
340
341     /*
342      * Find the total bytes in the message buffer. This is also the total
343      * number of bytes left to send being we haven't sent anything yet.
344      */
345     message_bytes_left_to_send = BYTE_COUNT(conn->outgoing_buffer_pointer);
346
347     /*
348      * Do a sanity check on the message size. If it's really big, assume that
349      * something BAD has happened in the upper level software and give up.
350      */
351     if (message_bytes_left_to_send > 2*MAX_REQUEST_SIZE) {
352         in_queue_message(ERROR_MSG, "message is too large(%lu) to send to the modalar", message_bytes_left_to_send);
353         return(FAILURE);
354     }
355
356     /*
357      * Kludge the "bytes left to send" as the first packet header is already
358      * included in the message that was passed down. This kludge is undone
359      * during the first pass through the while() loop when message_bytes_left_to_send
360

```

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM networkh/net_clnt.c	DATE 5/23/89 TIME 1:20:47 pm	PAGE # 4/7
LINE #	SOURCE TEXT		
361	/* is adjusted by P_HEADER_SIZE. Note that subsequent passes through the		
362	while() loop are O.K. as the packet header is overlaid onto the message data.		
363	*/		
364	message_bytes_left_to_send -= P_HEADER_SIZE;		
365			
366	while(message_bytes_left_to_send > 0) {		
367	/* Determine the size of the packet that we are going to send next */		
368	if (message_bytes_left_to_send > MAX_CLIENT_PACKET - P_HEADER_SIZE)		
369	packet_byte_count = MAX_CLIENT_PACKET;		
370	else packet_byte_count = message_bytes_left_to_send + P_HEADER_SIZE;		
371			
372	/* Set the appropriate byte count in the packet to be sent. */		
373	BYTE_COUNT(conn->outgoing_buffer_pointer) = packet_byte_count;		
374			
375	/* Send the message and wait for the acknowledgement. */		
376	if (send_single_message_get_ack(conn) != SUCCESS) {		
377	return(FAILURE);		
378	}		
379			
380	/* Adjust the buffer pointer to its new location, but back up enough		
381	to overlay a new packet header structure on the next sent packet. */		
382	/* NOTE: this trashes the message, but we don't care because the		
383	modeler has acknowledged its reception of the "trashed" data. */		
384	conn->outgoing_buffer_pointer += packet_byte_count - P_HEADER_SIZE;		
385			
386	/* We now have fewer message bytes left to send, adjust by decreasing the		
387	message byte count by the amount the buffer pointer was incremented. */		
388	message_bytes_left_to_send -= packet_byte_count - P_HEADER_SIZE;		
389			
390	}		
391			
392	return(SUCCESS);		
393			
394			
395	static u_short		
396	receive_multiple_message(conn)		
397	register CONNECTION *conn,		
398	{		
399	register char *ptr,		
400	register char *buffer_ptr,		
401			
402	register u_long packet_byte_count,		
403	register u_long message_bytes_left_to_receive,		
404			
405	PACKET_HEADER temporary; /* COPY */		
406			
407			
408			
409	/*		
410	* Note that upon entering this routine the incoming buffer already		
411	* contains the first part of the multiple message that is being received.		
412	*/		
413			
414	/*		
415	* Find the total bytes in the message buffer (i.e., how much we will receive).		
416	* The actual variable name is slightly misleading at this point due to the		
417	* fact that the first part of the message has already been received. Within		
418	* the while() loop below, the variable name has its true meaning.		
419	*/		
420	message_bytes_left_to_receive = PARTIAL_PACKET_COUNT(conn->incoming_buffer_base);		
421			
422	/*		
423	* Do a sanity check on the message size. If it's really big, assume that		
424	* something BAD has happened in the modeler software and give up.		
425	*/		
426	if (message_bytes_left_to_receive > 2*MAX_REPLY_SIZE) {		
427	in_queue_message(ERROR_MSG, "message is too large(!!!) to receive from the modeler", message_bytes_left_to_receive);		
428	return(FAILURE);		
429	}		
430			
431	/*		
432	* If the incoming message will have a total byte count that is larger		
433	* than the existing incoming buffer size, do a realloc() to the		
434	* eventual total size of the incoming message. A temporary pointer is		
435	* used to maintain the current buffer in case we are out of memory.		
436	*/		
437	if (message_bytes_left_to_receive+MAX_SERVER_PACKET > conn->incoming_buffer_size) {		
438	ptr = realloc(conn->incoming_buffer_base, (unsigned)(message_bytes_left_to_receive+MAX_SERVER_PACKET));		
439	if (ptr == (char *) NULL) {		
440	in_queue_message(SYS_ERROR_MSG, "out of memory on host for incoming messages");		
441	return(FAILURE);		
442	}		
443			
444	/* Update the connection data structure. */		
445	conn->incoming_buffer_base = ptr;		
446	conn->incoming_buffer_size = message_bytes_left_to_receive+MAX_SERVER_PACKET;		
447			
448	}		
449			
450	/* Initialize our local buffer pointer to the existing incoming buffer base.		
451	*/		
452	buffer_ptr = conn->incoming_buffer_base;		
453			
454	/*		
455	* Initialize the outgoing message buffer pointer.		
456	*/		
457	conn->outgoing_buffer_pointer = conn->outgoing_buffer_base;		
458			
459	/*		
460	* Do a structure assignment to set the current packet header as a		
461	* modeler acknowledgement for receiving the partial packet reply.		
462	*/		
463	*(PACKET_HEADER *) conn->outgoing_buffer_pointer = *(PACKET_HEADER *) buffer_ptr; /* COPY */		
464			
465	/*		
466	* Adjust the byte count in the acknowledgement.		
467	*/		
468	BYTE_COUNT(conn->outgoing_buffer_pointer) = P_HEADER_SIZE;		
469			
470	/*		
471	* Set the packet byte count to equal the size of the current packet.		
472	*/		
473	packet_byte_count = BYTE_COUNT(buffer_ptr);		
474			
475	/*		
476	* Adjust the buffer pointer to its new location, but back up enough to		
477	* overlay a new packet header structure for the next received packet.		
478	* NOTE: this will trash part of the incoming message, but we will		
479	* save and later restore that "trashed" data.		
480	*/		

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_clnt.c

DATE 5/23/89

PAGE #

TIME 1:20:47 pm

5/8

```

LINE # SOURCE TEXT
481 buffer_ptr += packet_byte_count - P_HEADER_SIZE,
482
483 /*
484  * We now have less message to receive. Adjust by decreasing the
485  * "bytes to receive" by the size of the incoming buffer.
486  */
487 message_bytes_left_to_receive -= packet_byte_count,
488
489 while(message_bytes_left_to_receive > 0) {
490     /* Do a structure assignment to save the last P_HEADER_SIZE
491     /* bytes so that they can be restored after the next incoming packet
492     /* is received and its header processed.
493     temporary = *(PACKET_HEADER *) buffer_ptr, /* COPY */
494
495     /* Set the incoming buffer pointer that send single_message() uses
496     /* to the current beginning of the whole incoming message buffer.
497     comm->incoming_buffer_pointer = buffer_ptr,
498
499     /* Send the request in a message and wait for the acknowledge.
500     if (send_single_message_get_ack(comm) != SUCCESS) {
501         return(FAILURE);
502     }
503
504     /* Do a structure assignment to set the current packet header as a
505     /* modular acknowledgement for receiving the partial packet reply.
506     *(PACKET_HEADER *) comm->outgoing_buffer_pointer = *(PACKET_HEADER *) buffer_ptr, /* COPY */
507
508     /* Adjust the byte count in the acknowledge.
509     BYTE_COUNT(comm->outgoing_buffer_pointer) = P_HEADER_SIZE,
510
511     /* Determine how many bytes are in the just received packet
512     packet_byte_count = BYTE_COUNT(buffer_ptr),
513
514     /* Restore the bytes that were saved above and trashed during the
515     /* message reception.
516     *(PACKET_HEADER *) buffer_ptr = temporary, /* COPY */
517
518     /* Adjust the buffer pointer to its new location, but back up enough to
519     /* overlay a new packet header structure for the next received packet.
520     /* NOTE: this will trash part of the incoming message, but we will
521     /* save and later restore that "trashed" data.
522     buffer_ptr += packet_byte_count - P_HEADER_SIZE,
523
524     /* We now have less message to receive, adjust by decreasing the
525     /* "bytes to receive" by the amount the buffer_ptr was incremented.
526     message_bytes_left_to_receive -= packet_byte_count - P_HEADER_SIZE,
527
528     }
529
530 /*
531  * Set the total number of bytes in the entire incoming message.
532  */
533 BYTE_COUNT(comm->incoming_buffer_base) = PARTIAL_PACKET_COUNT(comm->incoming_buffer_base),
534
535 return(SUCCESS);
536
537
538 static u_short
539 receive_packet_with_timeout(comm, milli_seconds)
540 register CONNECTION *comm,
541 register u_long milli_seconds,
542
543 int rts,
544 int readfds,
545 struct timeval timeout,
546 int counter;
547
548 #ifdef VMS
549 fd_set read_chans, write_chans, except_chans;
550 #endif
551
552 long elapsed_time,
553 long first_int_time = 0,
554
555 timeout.tv_sec = ((long) milli_seconds)/1000,
556 timeout.tv_usec = (((long) milli_seconds)%1000)*1000,
557 do_select:
558 readfds = 1 << comm->fd,
559 #ifdef VMS
560
561 timeout.tv_sec = 0,
562 timeout.tv_usec = 30,
563
564 counter = 0,
565 rts = 0,
566 while ((0 == rts) && ((milli_seconds/10) > counter)) {
567     FD_ZERO (&read_chans),
568     FD_ZERO (&write_chans),
569     FD_ZERO (&except_chans),
570     FD_SET (comm->fd, &read_chans),
571
572     counter++,
573
574     rts = select (0, &read_chans, &write_chans,
575     &except_chans, &timeout),
576
577     }
578
579 if (rts == -1) {
580     if ((rts = select(1 + comm->fd, &readfds,
581     (int *)NULL, (int *)NULL, &timeout)) == -1) {
582         #endif
583
584         /*
585          * If select() is interrupted, consider it a non-fatal error and
586          * go back to wait some more. To be completely correct we should
587          * not wait the same amount of time again, but rather decrease the
588          * time to wait. However, I don't know how much time to decrease
589          * by, so just start over again.
590          */
591         if (errno == EINTR) {
592             if (first_int_time == 0) {
593                 first_int_time = time(0),
594             } else {
595                 elapsed_time = time(0) - first_int_time,
596                 if (elapsed_time > (((long) milli_seconds)/1000)) {
597                     /* time is up */
598                     timeout.tv_sec = 0,
599
600

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
networkh/net_clnt.c

DATE 5/23/89
TIME 1:20:47 pm

PAGE #
6/9

```

LINE #          SOURCE TEXT
601          timeout.tv_usec = 0;
602          } else {
603              /* wait some more */
604              timeout.tv_sec = (((long) milli_seconds)/1000) + sleep_time;
605          }
606          /*
607           * get the select,
608           */
609          /*
610           * select() has failed, give up.
611           */
612          lm_queue_message(SYS_ERROR_MSG,
613              "Failure to select modular communication endpoint(socket)");
614          return(FAILURE);
615          }
616          if (rcv == 0) {
617              /* We have timed out.
618               * Return NOT_SUCCESS_OR_FAILURE.
619               */
620              return(NOT_SUCCESS_OR_FAILURE);
621          }
622          /* We have received the data. Note that it is
623           * not necessary to process the returned value
624           * as we know it is what we want. (there was
625           * some readfds selected for viewing).
626           */
627          /*
628           * Receive the acknowledge packet from the address in conn.
629           */
630          return(receive_packet(conn, conn->incoming_buffer_pointer, milli_seconds));
631      }
632      static u_short
633      receive_packet(conn, packet, milli_seconds)
634      register CONNECTION *conn,
635      register char *packet,
636      register u_long milli_seconds;
637      {
638          register int rcv;
639          register int byte_count;
640          register char *buffer_ptr;
641          #ifdef VMS
642              int sock_size = SOCK_SIZE;
643              register LM_HEADER *lm_header;
644          #endif
645          /*
646           * Initialize the receive buffer pointer, byte count and socket size.
647           */
648          byte_count = 0;
649          buffer_ptr = packet;
650          /*
651           * Loop until we receive all the packet, incrementing
652           * buffer_ptr and byte_count as we go.
653           */
654          do {
655              /*
656               * We use recv() rather than recvfrom() because performance is better.
657               * See the comments near the use of send() and connect().
658               */
659              rcv = recv(conn->fd, buffer_ptr, MAX_CLIENT_PACKET, 0);
660              if (rcv <= 0) {
661                  lm_queue_message(SYS_ERROR_MSG, "failure to receive any data from the modular");
662                  return(FAILURE);
663              }
664              byte_count += rcv;
665              buffer_ptr += rcv;
666              /* Don't stop until we have all the bytes.
667               */
668          } while (byte_count != BYTE_SWAP_LONG(BYTE_COUNT(packet)));
669          #ifdef VMS
670              /*
671               * This doesn't work if recvfrom() doesn't return all data in one call.
672               */
673          } while (byte_count != BYTE_COUNT(packet));
674          #endif
675          #ifdef VMS
676              /*
677               * Play games with the byte counting to convert it from modular order.
678               * The rest of the data will be converted as it is read from the packet.
679               */
680              lm_header = ((PACKET_HEADER *) packet)->lm_hdr;
681              lm_header->byte_count = BYTE_SWAP_LONG(lm_header->byte_count);
682              lm_header->process_id = BYTE_SWAP_LONG(lm_header->process_id);
683              lm_header->sequence_number = BYTE_SWAP_LONG(lm_header->sequence_number);
684              lm_header->partial_packet_count = BYTE_SWAP_LONG(lm_header->partial_packet_count);
685          #endif
686          return(SUCCESS);
687      }
688      static u_short
689      send_packet(conn, packet)
690      register CONNECTION *conn,
691      register char *packet,
692      {
693          register int rcv;
694          register int byte_count;
695          #ifdef VMS
696              int sock_size = SOCK_SIZE;
697              register LM_HEADER *lm_header;
698          #endif
699          /*
700           * We unconditionally send the packet to the connected user. Other routines
701           * determine whether the packet actually arrived at its destination.
702           */
703          /*
704           * Set the number of bytes we have to send.
705           */
706          byte_count = BYTE_COUNT(packet);
707          #ifdef VMS
708              /*
709               *
710               */
711          }
712          #endif
713      }

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkh/net_clnt.c	DATE 5/23/89	PAGE # 7/10
			TIME 1:20:47 pm	
LINE #	SOURCE TEXT			
721	/*			
722	* On VMS we must adjust the byte count of the packet being sent to account			
723	* for the fact that we add the UDP and IP headers in the "user" buffer.			
724	* This normalizes the byte count in the "net" to just the number of bytes			
725	* of real user data.			
726	BYTE_COUNT(packet) = sizeof(UDP_HEADER) + sizeof(IP_HEADER);			
727	/*			
728	* Play games with the byte counting to get it into modular required order.			
729	* The rest of the data was converted when it was put into the packet.			
730	/*			
731	lm_header = *((PACKET_HEADER *) packet) -> lm_hdr;			
732	lm_header->byte_count = BYTE_SNAP_LONG(lm_header->byte_count);			
733	lm_header->process_id = BYTE_SNAP_LONG(lm_header->process_id);			
734	lm_header->sequence_number = BYTE_SNAP_LONG(lm_header->sequence_number);			
735	lm_header->partial_packet_count = BYTE_SNAP_LONG(lm_header->partial_packet_count);			
736	endif			
737	/*			
738	* Increment our tracking of the total number of packets that are sent.			
739	* and send the packet to the address in conn.			
740	/*			
741	++lm_total_number_of_packets_sent;			
742	/*			
743	* The code is measurably faster if send() is used rather than sendto().			
744	* This requires the use of connect(), however. See comments elsewhere in			
745	* the code for details on connect()'s usage.			
746	/*			
747	scnt = send(conn->fd, packet, byte_count, 0);			
748	/*			
749	* Verify that all the packet bytes were sent.			
750	/*			
751	if (scnt != byte_count) {			
752	lm_queue_message(SYS_ERROR_MSG, "failure to send exactly %d bytes to the modular, sent %d instead", byte_count, scnt);			
753	return(FAILURE);			
754	}			
755	endif			
756	/*			
757	* Undo the byte counting that we done above. I'm not sure that			
758	* this is required, but it's safest to do it just the same.			
759	/*			
760	lm_header->byte_count = BYTE_SNAP_LONG(lm_header->byte_count);			
761	lm_header->process_id = BYTE_SNAP_LONG(lm_header->process_id);			
762	lm_header->sequence_number = BYTE_SNAP_LONG(lm_header->sequence_number);			
763	lm_header->partial_packet_count = BYTE_SNAP_LONG(lm_header->partial_packet_count);			
764	/*			
765	* Undo the adjustment that was done above (just in case anybody decides to			
766	* look at it in upper level routines).			
767	/*			
768	endif			
769	return(SUCCESS);			
770	}			
771	/*			
772	* short			
773	* in init_connection for client(conn)			
774	* register CONNECTION *conn,			
775	{			
776	u_long address, /* can't be made storage class register */			
777	register int fd,			
778	register int pid,			
779	conn->fd = -1, /* start with an invalidated conn structure */			
780	/*			
781	* WARNING: if you change this routine check reset_connection()			
782	* to see if the conn changes are needed there.			
783	/*			
784	* Verify that a name for the modular was specified.			
785	/*			
786	if (conn->name == (char *)NULL conn->name == "\0") {			
787	lm_queue_message(SYS_ERROR_MSG, "failure to specify a modular name");			
788	return(FAILURE);			
789	}			
790	/*			
791	* Create a datagram socket for communication.			
792	/*			
793	if ((fd=socket(AF_INET, SOCK_DGRAM, 0)) < 0) {			
794	lm_queue_message(SYS_ERROR_MSG, "could not create modular communication endpoint(socket)");			
795	return(FAILURE);			
796	}			
797	/*			
798	* Obvious?			
799	/*			
800	if (obtain_modular_address(fd, conn->name, address) != SUCCESS) {			
801	lm_close(fd);			
802	conn->fd = -1, /* invalidate the conn structure */			
803	return(FAILURE);			
804	}			
805	/*			
806	* Save the opened socket file descriptor, set the network family type			
807	* to internet, set the receiving port to be that of the modular, and			
808	* set the address to look for requests from any address.			
809	/*			
810	conn->fd = fd,			
811	conn->sock.family = AF_INET,			
812	conn->sock.port = MODULARS_ETHERNET_PORT,			
813	endif			
814	conn->sock.port = BYTE_SNAP_SHORT(MODULARS_ETHERNET_PORT),			
815	endif			
816	conn->sock.address = address;			
817	/*			
818	* Use connect() to permanently associate the previously set addresses			
819	* with the socket file descriptor. The use of this function is optional			
820	* given the use of sendto() and recvfrom() (rather than send() and recv()).			
821	* But the code is measurably faster if this association is done once, not			
822	* rather than every time sendto() or recvfrom() is called.			
823	/*			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_clnt.c

DATE 5/23/89

PAGE #

TIME 1:20:47 pm

8/11

```

LINE # SOURCE TEXT
841 if (connect(conn->fd, (struct sockaddr *)&conn->sock, SOCK_SIZE) != 0) {
842     in_queue_message(SYS_ERROR_MSG, "could not connect modular communication endpoint(socket)");
843     in_close(fd);
844     conn->fd = -1; /* invalidate the conn structure */
845     return(FAILURE);
846 }
847
848 /*
849  * Get the process id of the application. This number becomes
850  * part of the network protocol as an application ID.
851  */
852 pid = getpid();
853 if (pid < 0) {
854     in_queue_message(SYS_ERROR_MSG, "illegal host process id(fd)");
855     in_close(fd);
856     conn->fd = -1; /* invalidate the conn structure */
857     return(FAILURE);
858 }
859
860 /*
861  * Set the default values into the packet header construct that is
862  * used to track the state of the host communication.
863  */
864 conn->header.byte_count = -1;
865 conn->header.process_id = pid;
866 conn->header.sequence_number = 0;
867 conn->header.partial_packet_count = 0;
868
869 /*
870  * Initialize the partial packet indicators. (Technically, these don't
871  * need to be initialized for the client as the client server references
872  * them, but what the heck - let's be consistent and init everywhere.)
873  */
874 conn->am_sending = FALSE;
875 conn->bytes_left_to_send = 0;
876 conn->am_receiving = FALSE;
877 conn->bytes_left_to_receive = 0;
878
879 /*
880  * Set the incoming buffer time, end of current message,
881  * buffer size, and allocate memory space to receive messages.
882  */
883 conn->incoming_time = 0;
884 conn->incoming_buffer_size = 2*MAX_CLIENT_PACKET;
885 if ((conn->incoming_buffer_base = malloc(2*MAX_CLIENT_PACKET)) == (char *)NULL) {
886     in_queue_message(SYS_ERROR_MSG, "out of memory on host for incoming messages");
887     in_close(fd);
888     free(conn->incoming_buffer_base);
889     conn->fd = -1; /* invalidate the conn structure */
890     return(FAILURE);
891 }
892 conn->incoming_message_end = conn->incoming_buffer_base;
893
894 /*
895  * Initialize the incoming buffer pointer properly.
896  */
897 IN_RESET_INCOMING(conn);
898
899 /*
900  * Set the outgoing buffer time, end of buffer,
901  * buffer size, and allocate memory space to send messages.
902  */
903 conn->outgoing_time = 0;
904 conn->outgoing_buffer_size = 2*MAX_CLIENT_PACKET;
905 if ((conn->outgoing_buffer_base = malloc(2*MAX_CLIENT_PACKET)) == (char *)NULL) {
906     in_queue_message(SYS_ERROR_MSG, "out of memory on host for outgoing messages");
907     in_close(fd);
908     free(conn->incoming_buffer_base);
909     free(conn->outgoing_buffer_base);
910     conn->fd = -1; /* invalidate the conn structure */
911     return(FAILURE);
912 }
913 conn->outgoing_buffer_end = conn->outgoing_buffer_base + conn->outgoing_buffer_size;
914
915 /*
916  * Initialize the outgoing buffer pointer properly.
917  */
918 OUT_RESET_OUTGOING(conn);
919
920 return(SUCCESS);
921
922
923 static reset_network_connection (conn)
924 CONNECTION
925 *conn;
926 {
927     /*
928      * This routine completely resets the conn structure.
929      * Note that most of the code has been plagiarized from
930      * close_conn_for_client() and init_conn_for_client().
931      * If either of these routines change, check to see that
932      * this code does not need to be changed.
933      */
934     (void) free(conn->incoming_buffer_base);
935     (void) free(conn->outgoing_buffer_base);
936
937     /*
938      * Set the default values into the packet header construct that is
939      * used to track the state of the host communication.
940      */
941     conn->header.sequence_number = 0;
942     conn->header.partial_packet_count = 0;
943
944     /*
945      * Initialize the partial packet indicators. (Technically, these don't
946      * need to be initialized for the client as the client server references
947      * them, but what the heck - let's be consistent and init everywhere.)
948      */
949     conn->am_sending = FALSE;
950     conn->bytes_left_to_send = 0;
951     conn->am_receiving = FALSE;
952     conn->bytes_left_to_receive = 0;
953
954     /*
955      * Set the incoming buffer time, end of current message,

```

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM networkh/net_clnt.c	DATE	5/23/89	PAGE #
		TIME	1:20:47 pm	9/12

LINE #	SOURCE TEXT
1001	/* Buffer size, and allocate memory space to receive messages.
1002	*/
1003	conn->incoming_time = 0;
1004	conn->incoming_buffer_size = 2*MAX_CLIENT_PACKET;
1005	if ((conn->incoming_buffer_base = malloc(2*MAX_CLIENT_PACKET)) == (char *)NULL) {
1006	log_queue_message(SYS_ERROR_MSG, "out of memory on host for incoming messages");
1007	return(FAILURE);
1008	}
1009	conn->incoming_message_end = conn->incoming_buffer_base;
1010	/*
1011	Initialize the incoming buffer pointer properly.
1012	*/
1013	LM_RESET_INCOMING(conn);
1014	/*
1015	Set the outgoing buffer time, end of buffer,
1016	buffer size, and allocate memory space to send messages.
1017	*/
1018	conn->outgoing_time = 0;
1019	conn->outgoing_buffer_size = 2*MAX_CLIENT_PACKET;
1020	if ((conn->outgoing_buffer_base = malloc(2*MAX_CLIENT_PACKET)) == (char *)NULL) {
1021	log_queue_message(SYS_ERROR_MSG, "out of memory on host for outgoing messages");
1022	return(FAILURE);
1023	}
1024	conn->outgoing_buffer_end = conn->outgoing_buffer_base + conn->outgoing_buffer_size;
1025	/*
1026	Initialize the outgoing buffer pointer properly.
1027	*/
1028	LM_RESET_OUTGOING(conn);
1029	return(SUCCESS);
1030	}
1031	u_short
1032	lm_close_connection_for_client(conn)
1033	register CONNECTION *conn;
1034	{
1035	/*
1036	WARNING: If you change this routine check recent network_communic.c
1037	to see if the same changes are needed there.
1038	*/
1039	if (conn->id == -1) {
1040	/* This conn has already been closed */
1041	return(SUCCESS);
1042	}
1043	if (lm_close(conn->id) != 0) {
1044	log_queue_message(SYS_ERROR_MSG, "failure to close nodular communication endpoint(socket)");
1045	return(FAILURE);
1046	}
1047	(void) free(conn->incoming_buffer_base);
1048	(void) free(conn->outgoing_buffer_base);
1049	conn->id = -1; /* Invalidate the conn structure */
1050	return(SUCCESS);
1051	}
1052	static u_short
1053	get_is_nodular_address(id, nodular_name, address)
1054	int id,
1055	char *nodular_name,
1056	u_long *address,
1057	{
1058	register struct hostent *hp;
1059	return struct hostent *gethostbyname();
1060	/*
1061	Use the standard UNIX gethostbyname() routine for getting
1062	the network address of the nodular.
1063	*/
1064	hp = gethostbyname(nodular_name);
1065	if (hp == (struct hostent *)NULL) {
1066	log_queue_message(ERROR_MSG, "nodular '%s' is unknown to the hosts database", nodular_name);
1067	return(FAILURE);
1068	}
1069	/*
1070	Return the found address in the appropriate place.
1071	*/
1072	*address = *(u_long *) hp->h_addr;
1073	return(SUCCESS);
1074	}
1075	static u_short
1076	process_timeout(number_of_message_attempts, network_timeout)
1077	register u_short *number_of_message_attempts,
1078	register u_long *network_timeout,
1079	{
1080	/*
1081	Track the total number of retries that have been attempted
1082	since this process began.
1083	*/
1084	++lm_total_number_of_msg_retries; /* global */
1085	/*
1086	Obviously, something has gone wrong if we get into this routine.
1087	Let's be safe and print a warning message.
1088	*/
1089	/*
1090	Update, people don't want to see warning messages if something
1091	isn't really wrong. So for now we'll not say anything.
1092	*/
1093	/*
1094	Increment the number of times that we have tried to process
1095	the current message. If it has been too many times, give up.
1096	*/
1097	if (++(*number_of_message_attempts) >= lm_max_timeout_retry_attempts) {
1098	log_queue_message(ERROR_MSG, "nodular not responding");
1099	return(FAILURE);
1100	}

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkh/net_clnt.c	DATE 5/23/89	PAGE # 10/13
LINE #		SOURCE TEXT		
1081		}		
1082		/*		
1083		Obvious: something has gone wrong if we get into this routine.		
1084		Let's be safe and print a warning message at infrequent intervals.		
1085		*/		
1086		if (((++lm_total_number_of_timeouts & lm_network_warning_freq) == 0) {		
1087		lm_queue_message(WARNING_MSG, "All out of 1lu network packets have timed out, is your network OK?",		
1088		lm_total_number_of_timeouts,		
1089		lm_total_number_of_packets_sent		
1090);		
1091		}		
1092		/*		
1093		Increment the time that we wait for the next acknowledgement.		
1094		This is a simple attempt at adjusting for modeler load, and		
1095		is just a simple increase in wait time for each timeout.		
1096		*/		
1097		network_timeout += lm_network_timeout;		
1098		/*		
1099		Try, try, again.		
1100		*/		
1101		return(SUCCESS);		
1102		}		
1103		}		
1104		static u_short		
1105		process_mismatch()		
1106		{		
1107		/*		
1108		Track the total number of retries that have been attempted		
1109		since this process began.		
1110		*/		
1111		++lm_total_number_of_msg_retries; /* global */		
1112		/*		
1113		Obviously, something has gone wrong if we get into this routine.		
1114		Let's be safe and print a warning message at infrequent intervals.		
1115		*/		
1116		if (((++lm_total_number_of_mismatches & lm_network_warning_freq) == 0) {		
1117		lm_queue_message(WARNING_MSG, "All out of 1lu network packets have been mismatched, is your network OK?",		
1118		lm_total_number_of_mismatches,		
1119		lm_total_number_of_packets_sent		
1120);		
1121		}		
1122		/*		
1123		Try, try, again.		
1124		*/		
1125		return(SUCCESS);		
1126		}		
1127		}		
1128		#define SMALL_BUFFER 256		
1129		/*		
1130		lm_is_modeler_alive() is guaranteed to determine if the modeler is alive		
1131		or dead (defined below) is less than 3 seconds.		
1132		*/		
1133		It returns LM_ERROR if some internal error occurred. (Make sure you		
1134		read what the LM_WARNING return code means.)		
1135		*/		
1136		It returns LM_SUCCESS if everything went OK and the modeler is alive.		
1137		This is the only return code for which "state" contains valid information.		
1138		*/		
1139		It returns LM_WARNING if everything went OK except for getting a response		
1140		from the modeler. I wanted to differentiate all out of my control errors		
1141		from the condition that the modeler is not alive. This is done by looking		
1142		at the return code rather than doing some special garbage with "state".		
1143		*/		
1144		In order for this routine to return LM_SUCCESS (i.e., say that the modeler		
1145		is alive), it must be able to create a socket and send and receive packets		
1146		on the host - AND - the modeler must be up and running to the point where		
1147		it is correctly processing LAMCA interrupts.		
1148		*/		
1149		lm_is_modeler_alive(modeler_name, state)		
1150		char *modeler_name,		
1151		char *state,		
1152		{		
1153		return send_on_special_port(modeler_name, state,		
1154		MODELERS_ARE_YOU_ALIVE_PORT, AXA_INQUIRE, 0, 0);		
1155		}		
1156		lm_request_interrupt(modeler_name, state)		
1157		char *modeler_name,		
1158		char *state,		
1159		{		
1160		return send_on_special_port(modeler_name, state,		
1161		MODELERS_ARE_YOU_ALIVE_PORT, AXA_INTERRUPT, 0, 0);		
1162		}		
1163		lm_remote_reset(modeler_name, state)		
1164		char *modeler_name,		
1165		char *state,		
1166		{		
1167		return send_on_special_port(modeler_name, state,		
1168		MODELERS_DEATH_PORT, AXA_KILL, 0, 0);		
1169		}		
1170		static send_on_special_port(modeler_name, state, port, cmd, size, buffer)		
1171		char *modeler_name,		
1172		char *state,		
1173		char *buffer,		
1174		{		
1175		register u_short rts;		
1176		int fd;		
1177		u_long address;		
1178		u_short number_of_message_attempts; /* can't be made static register */		
1179		{		
1180		CONNECTION conn;		
1181		{		
1182		char in_buffer[SMALL_BUFFER];		
1183		char out_buffer[512 + sizeof(LM_HEADER) + 8];		
1184		long		
1185		int packet_size = size + sizeof(LM_HEADER) + 8;		
1186		{		
1187		*state = 0;		
1188		}		
1189		}		
1190		}		
1191		}		
1192		}		
1193		}		
1194		}		
1195		}		
1196		}		
1197		}		
1198		}		
1199		}		
1200		}		

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkh/net_clnt.c

DATE

5/23/89

PAGE #

TIME

1:20:47 pm

11/14

```

1201 if (packet_size > sizeof(out_buffer)) {
1202     lm_queue_message(ERROR_MSG,
1203         "special packet size too big (%d bytes)", packet_size);
1204     return(LM_ERROR);
1205 }
1206
1207 /* Verify that a name for the modeler was specified */
1208 if (modeler_name == (char *)NULL || (modeler_name == "\0")) {
1209     lm_queue_message(ERROR_MSG, "failure to specify a modeler name");
1210     return(LM_ERROR);
1211 }
1212
1213 /* Store the name away in the case for lower level routine's processing */
1214 conn.name = modeler_name;
1215
1216 /* Create a socket for communication */
1217 if ((socket(AF_INET, SOCK_STREAM, 0)) < 0) {
1218     lm_queue_message(SYS_ERROR_MSG, "could not create modeler communication endpoint(socket)");
1219     return(LM_ERROR);
1220 }
1221
1222 if (obtain_modelers_address(id, conn.name, &address) != SUCCESS) {
1223     (void)lm_close(fd);
1224     return(LM_ERROR);
1225 }
1226
1227 conn.fd = fd;
1228 conn.sock.family = AF_INET;
1229 conn.sock.port = htons(port);
1230 conn.sock.address = address;
1231
1232 if (connect(conn.fd, (struct sockaddr *)&conn.sock, SOCK_SIZE) != 0) {
1233     (void)lm_close(fd);
1234     lm_queue_message(SYS_ERROR_MSG, "could not connect modeler communication endpoint(socket)");
1235     return(LM_ERROR);
1236 }
1237
1238 conn.incoming_buffer_base = in_buffer;
1239 conn.incoming_buffer_pointer = in_buffer;
1240 conn.outgoing_buffer_base = out_buffer;
1241 conn.outgoing_buffer_pointer = out_buffer;
1242
1243 /* We really don't have an outgoing message, so just align 4 bytes in */
1244 lptr = (long *)&conn.outgoing_buffer_base + sizeof(LM_HEADER);
1245 lptr += 4;
1246 lptr += size;
1247 if (size > 0) {
1248     bcopy(buffer, (char *)lptr, size);
1249 }
1250 BYTE_COUNT(conn.outgoing_buffer_base) = packet_size;
1251
1252 /* It doesn't matter what these fields are, but set them to 0 anyway */
1253 /* There is some small probability that they could find their way into */
1254 /* higher level code and cause serious corruptions if not caught */
1255 PROCESS_ID(conn.outgoing_buffer_base) = 0;
1256 SEQUENCE_NUMBER(conn.outgoing_buffer_base) = 0;
1257 PARTIAL_PACKET_COUNT(conn.outgoing_buffer_base) = 0;
1258
1259 number_of_message_attempts = 0;
1260
1261 try_try_again:
1262 if (send_packet(conn, conn.outgoing_buffer_base) != SUCCESS) {
1263     (void)lm_close(fd);
1264     return(LM_ERROR);
1265 }
1266
1267 rts = receive_packet_with_timeout(conn, (u_long)500); /* 0.5 seconds */
1268 if (rts == SUCCESS) {
1269     /* Fall through to below to process the received packet */
1270 }
1271 else if (rts == NOT_SUCCESS_OR_FAILURE) {
1272     /* We have timed out and must retransmit the request */
1273     if (++number_of_message_attempts >= 3) {
1274         (void)lm_close(fd);
1275         lm_queue_message(WARNING_MSG, "modeler does not respond");
1276         return(LM_WARNING);
1277     }
1278     goto try_try_again;
1279 }
1280 else {
1281     (void)lm_close(fd);
1282     return(LM_ERROR);
1283 }
1284
1285 /* Ignore version for now, just return the state */
1286 version = conn.incoming_buffer_base[0+sizeof(PACKET_HEADER)];
1287
1288 *state = conn.incoming_buffer_base[1+sizeof(PACKET_HEADER)];
1289 (void)lm_close(fd);
1290
1291 if (*state == RUNNING_LOOPMODE) {
1292     lm_queue_message(WARNING_MSG, "modeler running loop mode");
1293     return(LM_WARNING);
1294 }
1295
1296 return(LM_SUCCESS);
1297
1298 static
1299 lm_close(fd)
1300 int fd;
1301 {
1302     if (fd != VMS)
1303         return( netclose(fd) );
1304     else
1305         return( close(fd) );
1306 }
1307 #endif
1308
1309

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkm/net_srv.c

DATE 5/23/89

PAGE #

TIME 4:42:06 pm

1/1

```

1  /* SOCS ID: net_srv.c rev 1.5, 5/9/89 at 17:28:38 */
2  #include "common.h"
3  #include "message.h"
4  #include "lance.h"
5  #include "network.h"
6  #include "task.h"
7  #include "time.h"
8  #include "cpu.h"
9  #include "msg_err.h"
10 #include "svr_err.h"
11 #include "vrtx.h"
12
13 extern u_short handle_first_part_of_reply();
14 extern u_short send_next_part_of_reply();
15 extern u_short handle_first_part_of_request();
16 extern u_short receive_next_part_of_request();
17
18 extern u_short set_up_out_of_luck_user();
19 extern u_short set_close_connection_for_server();
20 extern u_short receive_packet();
21 extern u_short send_packet();
22 extern u_short send_packet_to_host();
23 extern u_short find_or_create_a_connection();
24 extern u_short init_connection_for_server();
25 extern u_short close_connection_for_server();
26
27 u_long lm_dead_user_mask = 0;
28
29 #define vprintf (void)printf
30
31 #define MSGLEN 50
32 static char tx_buffer[MAX_USERS] [ sizeof(PACKET_HEADER) + MSGLEN];
33 static char *tx_buffer_ptr;
34
35 #define LM_RESET_RAW_TX_BUFFER(X) (tx_buffer_ptr = tx_buffer[ (X) ] + sizeof(PACKET_HEADER) - sizeof(LM_HEADER) )
36 #define LM_PUT_CHAR_TX_BUFFER(X) (*tx_buffer_ptr++ = (X))
37 #define LM_PUT_SHORT_TX_BUFFER(X) (*(u_short *)tx_buffer_ptr++ = (X))
38 #define LM_PUT_LONG_TX_BUFFER(X) (*(u_long *)tx_buffer_ptr++ = (X))
39 #define LM_TX_BYTECOUNT(X) (tx_buffer_ptr - tx_buffer[(X)] - P_HEADER_SIZE + LM_HEADER_SIZE)
40
41 #ifdef MODLER
42 extern BOOT_STRUCT boot;
43 #else
44 BOOT_STRUCT boot;
45 #endif
46 static u_char active_users = 0;
47
48 static CONNECTION base_conn;
49
50 u_long network_timeout;
51 u_long print_sequence_mismatches = FALSE;
52
53 /*
54  * LMSig: LMSX
55  * LMSI: live
56  * LMSD: dead
57  * LMSK: kill
58  */
59 #define LMSig 0x04d5349
60 #define LMSIlive 0x0c97865
61 #define LMSIdad 0x0456184
62 #define LMSIkill 0x0b696c0
63 #define UNWANTED_USERS 1
64
65 static char dump_area[ MAX_SERVER_PACKET ];
66
67 CONNECTION *table_of_conns[MAX_USERS+UNWANTED_USERS];
68
69 u_short
70 lm_send_reply(conn)
71 register CONNECTION *conn,
72 {
73     u_short status;
74
75     conn->as_processing = FALSE;
76
77     if (conn->as_sending == FALSE) {
78         /* Copy the request packet sequence number to the reply packet header */
79         SEQUENCE_NUMBER(conn->outgoing_buffer_base) = SEQUENCE_NUMBER(conn->incoming_buffer_base);
80
81         /* Set the total number of bytes in the outgoing message buffer */
82         /* It is absolutely critical that this be done here, before any */
83         /* further buffer processing is done as the latter processing */
84         /* changes the value of conn->outgoing_buffer_pointer */
85         BYTE_COUNT(conn->outgoing_buffer_base) = (u_long)conn->outgoing_buffer_pointer - (u_long)conn->outgoing_buffer_base - P_HEADER_SIZE + LM_HEADER_SIZE;
86
87         if (BYTE_COUNT(conn->outgoing_buffer_base) <= MAX_CLIENT_PACKET) {
88             /* The reply WILL fit into a single network packet. Set the */
89             /* indication that this is the only packet for this request. */
90             conn->header.partial_packet_count = 0;
91             conn->last_outgoing_buffer_pointer = conn->outgoing_buffer_base;
92             if (send_packet(conn, conn->outgoing_buffer_base) != SUCCESS) {
93                 return(FAILURE);
94             }
95         }
96         else {
97             /* The reply will NOT fit into a single network packet. */
98             /* Set the indication that this is a partial packet. */
99             conn->header.partial_packet_count = BYTE_COUNT(conn->outgoing_buffer_base);
100
101             if (handle_first_part_of_reply(conn) != SUCCESS) {
102                 return(FAILURE);
103             }
104         }
105
106         conn->as_timing_out = TRUE;
107         conn->time_to_live = (lm_tick + 5) + network_timeout;
108         conn->time_at_last_response = lm_tick;
109     }
110
111     else {
112         /* We are sending a partial reply. The first part of the */
113         /* reply has already been sent and is present in */
114         /* conn->outgoing_buffer_base. Go put the next */
115         /* part of the reply. This may or may not be the */
116         /* last part of the total reply. It's checked below. */
117         if ((status = send_next_part_of_reply(conn)) != SUCCESS) {
118             conn->as_timing_out = TRUE;
119             conn->time_to_live = (lm_tick + 5) + network_timeout;
120         }
121     }
122 }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
networkm/net_srv.c

DATE 5/23/89
TIME 4:42:06 pm

PAGE #
2/2

```

LINE # SOURCE TEXT
120      conn->time_of_last_response = 1u_tick,
121      }
122      return( status );
123      }
124      return(SUCCESS);
125      }
126
127 static u_short
128 handle_first_part_of_reply(conn)
129 register CONNECTION *conn,
130 {
131     register u_long total_bytes_to_send;
132
133     /*
134      * Set the total number of bytes that must be sent.
135      */
136     total_bytes_to_send = BYTE_COUNT(conn->outgoing_buffer_base);
137
138     /*
139      * Do a sanity check on the buffer size. If it's really big, assume
140      * that something BAD has happened in the modular software and give up.
141      */
142     if (total_bytes_to_send > 2*MAX_REPLY_SIZE) {
143         in_queue_message(ERROR_MSG, "message is too large(!!) to send to the host", total_bytes_to_send);
144         return(FAILURE);
145     }
146
147     /*
148      * Set the partial reply send indicator.
149      */
150     conn->am_sending = TRUE;
151
152     /*
153      * Adjust the byte count in the first part of the reply.
154      */
155     BYTE_COUNT(conn->outgoing_buffer_base) = MAX_CLIENT_PACKET;
156
157     /*
158      * Send the first part of the reply to the address in conn.
159      */
160     conn->last_outgoing_buffer_pointer = conn->outgoing_buffer_base;
161     if (send_packet(conn, conn->outgoing_buffer_base) != SUCCESS) {
162         return(FAILURE);
163     }
164
165     /*
166      * Set the number of bytes that we have left to send.
167      */
168     conn->bytes_left_to_send = total_bytes_to_send - MAX_CLIENT_PACKET;
169
170     /*
171      * Set the outgoing buffer pointer to the beginning of
172      * the sent part of the reply that will be sent.
173      */
174     conn->outgoing_buffer_pointer = conn->outgoing_buffer_base + MAX_CLIENT_PACKET + P_HEADER_SIZE - 1u_HEADER_SIZE;
175     return(SUCCESS);
176 }
177
178 static u_short
179 send_next_part_of_reply(conn)
180 register CONNECTION *conn,
181 {
182     register u_long packet_byte_count;
183
184     /*
185      * Note that upon entering this routine the first part of the multiple
186      * message reply has already been sent and is present in the outgoing buffer.
187      */
188
189     /*
190      * Receive the request packet from the address in conn.
191      * This is an acknowledgment of the previously sent partial reply
192      * and also a request to receive the next part of the reply.
193      */
194
195     if (receive_packet(conn, conn->incoming_buffer_base) != SUCCESS) {
196         return(FAILURE);
197     }
198
199     if (conn->header.sequence_number != SEQUENCE_NUMBER(conn->incoming_buffer_base)) {
200
201         if (print_sequence_mismatches == TRUE) {
202             vprintf("MSG1: %d, %d, %d\n",
203                     conn->id,
204                     SEQUENCE_NUMBER(conn->incoming_buffer_base),
205                     conn->header.sequence_number);
206         }
207
208         if (conn->header.sequence_number == SEQUENCE_NUMBER(conn->incoming_buffer_base)+1) {
209             if (conn->am_processing == FALSE) {
210                 if (send_packet(conn, conn->last_outgoing_buffer_pointer) != SUCCESS) {
211                     return(FAILURE);
212                 }
213             }
214             return(SUCCESS);
215         }
216         else {
217             return(FAILURE);
218         }
219     }
220
221     conn->header.sequence_number++;
222
223     /*
224      * Back up the current outgoing buffer pointer enough to allow us
225      * to overlay a new packet header structure for the next sent packet.
226      * NOTE: this will trash part of the outgoing message, but we don't
227      * care because the host has already acknowledged its reception.
228      */
229     conn->outgoing_buffer_pointer -= P_HEADER_SIZE;
230
231     /*
232      * Copy the request packet sequence number to the reply packet header.
233      */
234 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkm/net_srv.c

DATE

5/23/89

PAGE #

TIME

4:42:06 pm

3/3

```

LINE # SOURCE TEXT
240 SEQUENCE_NUMBER(coas->outgoing_buffer_pointer) = SEQUENCE_NUMBER(coas->incoming_buffer_base);
241
242 /*
243  * Determine the size of the packet that we are going to send next.
244  */
245
246 if (coas->bytes_left_to_send > MAX_CLIENT_PACKET - LM_HEADER_SIZE)
247     packet_byte_count = MAX_CLIENT_PACKET;
248 else packet_byte_count = coas->bytes_left_to_send + LM_HEADER_SIZE;
249
250 /*
251  * Set the appropriate byte count in the packet to be sent.
252  */
253 BYTE_COUNT(coas->outgoing_buffer_pointer) = packet_byte_count;
254
255 /*
256  * Send the next part of the reply to the address in coas.
257  */
258 if (send_packet(coas, coas->outgoing_buffer_pointer) != SUCCESS) {
259     return(FAILURE);
260 }
261
262 /*
263  * Set the number of bytes that we have left to send.
264  */
265 coas->bytes_left_to_send -= packet_byte_count - LM_HEADER_SIZE;
266
267 /*
268  * Set the outgoing buffer pointer to the beginning of
269  * the next part of the reply that will be sent.
270  */
271 coas->last_outgoing_buffer_pointer = coas->outgoing_buffer_pointer;
272 coas->outgoing_buffer_pointer += packet_byte_count + P_HEADER_SIZE - LM_HEADER_SIZE;
273
274 /*
275  * Check for having sent all of the reply.
276  */
277 if (coas->bytes_left_to_send == 0) {
278     /* Reset the currently sending reply flag. */
279     coas->am_sending = FALSE;
280 }
281
282 return(SUCCESS);
283
284
285
286 u_short
287 lm_receive_request(coas)
288 register CONNECTION *coas,
289 {
290     if (coas->am_receiving == FALSE) {
291         /*
292          * If the node has received the whole request but is
293          * still working on the reply then we should throw away
294          * all packets coming in from the client and make sure
295          * that we don't overwrite the incoming buffer.
296          */
297         if (coas->am_processing == TRUE) {
298             if (receive_packet(coas, dump_area) != SUCCESS) {
299                 return(FAILURE);
300             }
301             return(SUCCESS);
302         }
303         /*
304          * Receive the request packet from the address in coas.
305          */
306         if (receive_packet(coas, coas->incoming_buffer_base) != SUCCESS) {
307             return(FAILURE);
308         }
309
310         if (SEQUENCE_NUMBER(coas->incoming_buffer_base) != coas->header.sequence_number) {
311             if (print_sequence_mismatches == TRUE) {
312                 fprintf(stderr, "Sequence mismatch: %d, %d, %d\n",
313                     coas->id,
314                     SEQUENCE_NUMBER(coas->incoming_buffer_base),
315                     coas->header.sequence_number);
316             }
317             if (SEQUENCE_NUMBER(coas->incoming_buffer_base) + 1 == coas->header.sequence_number) {
318                 if (coas->am_processing == FALSE) {
319                     /* (PACKET_HEADER *) coas->incoming_buffer_base =
320                      * (PACKET_HEADER *) coas->outgoing_buffer_base;
321                     BYTE_COUNT(coas->incoming_buffer_base) = sizeof(LM_HEADER);
322                     if (send_packet(coas, coas->last_outgoing_buffer_pointer) != SUCCESS)
323                         return(FAILURE);
324                     return(SUCCESS);
325                 }
326                 return(SUCCESS);
327             }
328             else {
329                 return(FAILURE);
330             }
331         }
332         coas->header.sequence_number++;
333         /*
334          * Overlay a packet header on the request to check for a partial request.
335          */
336         if (PARTIAL_PACKET_COUNT(coas->incoming_buffer_base) != 0) {
337             /* We have not received all of the request. Set up,
338              * check, and acknowledge the first part of the request.
339              */
340             if (handle_first_part_of_request(coas) != SUCCESS) {
341                 return(FAILURE);
342             }
343             return(PARTIAL_REQUEST);
344         }
345         /* Properly set up the incoming and outgoing
346          * buffers for the upper software layers.
347          */
348         LM_RESET_INCOMING(coas);
349         LM_RESET_OUTGOING(coas);
350         LM_SET_END_INCOMING(coas);
351         return(FULL_REQUEST);
352     }
353 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

networkm/net_srv.c

DATE 5/23/89

PAGE #

TIME 4:42:06 pm

4/4

```

LINE # SOURCE TEXT
340 else {
341     /* Am receiving a partial request. The first part of
342     /* the request has already been received and is present
343     /* in conn->incoming_buffer_base. Go get the next
344     /* part of the request. This may or may not be the
345     /* last part of the total request. It's checked below
346     */
347     return(receive_next_part_of_request(conn));
348 }
349
350 static u_short
351 handle_first_part_of_request(conn)
352 register CONNECTION *conn;
353 {
354     register char *ptr;
355     register u_long total_bytes_to_receive;
356
357     /* Set the total number of bytes that will be received.
358     */
359     total_bytes_to_receive = PARTIAL_PACKET_COUNT(conn->incoming_buffer_base);
360
361     /* Do a sanity check on the buffer size. If it's really big, assume
362     /* that something BAD has happened in the host software and give up.
363     */
364     if (total_bytes_to_receive > 2*MAX_REQUEST_SIZE) {
365         in_queue_message(ERROR_MSG, "message is too large(8lu) to receive from the host", total_bytes_to_receive);
366         return(FAILURE);
367     }
368
369     /* If the incoming message will have a total byte count that is larger
370     /* than the existing incoming buffer size. Do a realloc() to the
371     /* eventual total size of the incoming message. A temporary pointer is
372     /* used to maintain the current buffer in case we are out of memory.
373     */
374     if (total_bytes_to_receive + MAX_SERVER_PACKET > conn->incoming_buffer_size) {
375         ptr = realloc(conn->incoming_buffer_base,
376                     (unsigned)total_bytes_to_receive + MAX_SERVER_PACKET);
377         if (ptr == (char *) NULL) {
378             in_queue_message(SYS_ERROR_MSG, "out of memory on modeler for incoming messages");
379             return(FAILURE);
380         }
381         /* Update the connection data structure.
382         */
383         conn->incoming_buffer_base = ptr;
384         conn->incoming_buffer_size = total_bytes_to_receive + MAX_SERVER_PACKET;
385     }
386
387     /* Set the partial request receive indicator.
388     */
389     conn->am_receiving = TRUE;
390     conn->am_timing_out = TRUE;
391     conn->time_to_live = (in_tick + 5) * network_timeout;
392
393     /* Set the number of bytes that we have left to receive.
394     */
395     conn->bytes_left_to_receive = total_bytes_to_receive - BYTE_COUNT(conn->incoming_buffer_base);
396
397     /* Set the incoming buffer pointer to the end of the just received partial request.
398     */
399     conn->incoming_buffer_pointer = conn->incoming_buffer_base + BYTE_COUNT(conn->incoming_buffer_base) + P_HEADER_SIZE - LM_HEADER_SIZE;
400
401     /* Do a structure assignment to set the current packet header as a
402     /* modular acknowledgement for receiving the partial packet request.
403     */
404     /* FIRST: too much is getting copied...
405     /* (PACKET_HEADER *) conn->outgoing_buffer_base = (PACKET_HEADER *) conn->incoming_buffer_base; /* COPY */
406
407     /* Adjust the byte count in the acknowledge.
408     */
409     BYTE_COUNT(conn->outgoing_buffer_base) = LM_HEADER_SIZE;
410
411     /* Send the acknowledgement to the address in conn.
412     */
413     if (send_packet(conn, conn->outgoing_buffer_base) != SUCCESS) {
414         return(FAILURE);
415     }
416
417     return(SUCCESS);
418 }
419
420 static u_short
421 receive_next_part_of_request(conn)
422 register CONNECTION *conn;
423 {
424     register char *buffer_ptr;
425     PACKET_HEADER temporary; /* COPY */
426
427     /* Note that upon entering this routine the incoming buffer already
428     /* contains the first part of the multiple message that is being received.
429     */
430
431     /* Initialize our local buffer pointer to the existing incoming buffer
432     /* pointer, but back up enough to allow us to overlay a new packet
433     /* header structure for the next received packet.
434     /* NOTE: this will trash part of the incoming message, but we
435     /* will save and later restore that "crashed" data.
436     */
437     buffer_ptr = conn->incoming_buffer_pointer - P_HEADER_SIZE;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
networkm/net_srv.c

DATE 5/23/89
TIME 4:42:06 pm

PAGE #
5/5

```

LINE # SOURCE TEXT
480
481 /*
482  * Do a structure assignment to save the last P_HEADER_SIZE
483  * bytes so that they can be restored after the next incoming packet
484  * is received and its header processed.
485  */
486 temporary = *(PACKET_HEADER *) buffer_ptr; /* COPY */
487
488 /*
489  * Receive the request packet from the address in conn.
490  */
491 if (receive_packet(conn, buffer_ptr) != SUCCESS) {
492     return(FAILURE);
493 }
494
495 /* Check the incoming packet sequence number if it is a duplicate packet */
496 /* generate a new reply */
497
498 if (SEQUENCE_NUMBER(buffer_ptr) != conn->header.sequence_number) {
499     if (print_sequence_mismatch == TRUE) {
500         fprintf(stderr, "seq mismatch: %d, %d, %d\n",
501             conn->id,
502             SEQUENCE_NUMBER(buffer_ptr),
503             conn->header.sequence_number);
504     }
505     if (SEQUENCE_NUMBER(buffer_ptr) + 1 == conn->header.sequence_number) {
506         if (conn->am_processing == FALSE) {
507             *(PACKET_HEADER *) conn->outgoing_buffer_base = *(PACKET_HEADER *) buffer_ptr;
508             BYTE_COUNT(conn->outgoing_buffer_base) = LM_HEADER_SIZE;
509             *(PACKET_HEADER *) buffer_ptr = temporary;
510             if (send_packet(conn, conn->outgoing_buffer_base) != SUCCESS)
511                 return(FAILURE);
512             conn->am_timing_out = TRUE;
513             conn->time_to_live = (lm_tick * 5) + network_timeout;
514             return(PARTIAL_REQUEST);
515         } else {
516             return(FAILURE);
517         }
518     }
519 }
520
521 conn->header.sequence_number++;
522
523 /*
524  * Set the number of bytes that we have left to receive.
525  */
526 conn->bytes_left_to_receive -= BYTE_COUNT(buffer_ptr) - LM_HEADER_SIZE;
527
528 /* Check for having received all of the request.
529  */
530 if (conn->bytes_left_to_receive == 0) {
531     /* generate a new reply */
532
533     /*
534      * Reset the am currently receiving request flag.
535      */
536     conn->am_receiving = FALSE;
537
538     /*
539      * Set the total number of bytes in the entire incoming message.
540      */
541     BYTE_COUNT(conn->incoming_buffer_base) = PARTIAL_PACKET_COUNT(conn->incoming_buffer_base);
542     SEQUENCE_NUMBER(conn->incoming_buffer_base) = SEQUENCE_NUMBER(buffer_ptr);
543
544     /*
545      * Restore the bytes that were saved above and
546      * trashed during the message reception.
547      */
548     *(PACKET_HEADER *) buffer_ptr = temporary; /* COPY */
549
550     /*
551      * Properly set up the incoming and outgoing
552      * buffers for the upper software layers.
553      */
554     LM_RESET_INCOMING(conn);
555     LM_RESET_OUTGOING(conn);
556     LM_SET_END_INCOMING(conn);
557     return(FULL_REQUEST);
558 }
559
560 /*
561  * Set the incoming buffer pointer to the end of the just received partial request.
562  */
563 conn->incoming_buffer_pointer += BYTE_COUNT(buffer_ptr) - LM_HEADER_SIZE;
564
565 /*
566  * Do a structure assignment to save the current packet header as a
567  * modular acknowledgement for receiving the partial packet reply.
568  */
569 /* FIRST, too much is getting copied. */
570 *(PACKET_HEADER *) conn->outgoing_buffer_base = *(PACKET_HEADER *) buffer_ptr; /* COPY */
571
572 /*
573  * Adjust the byte count in the acknowledgement.
574  */
575 BYTE_COUNT(conn->outgoing_buffer_base) = LM_HEADER_SIZE;
576
577 /*
578  * Restore the bytes that were saved above and
579  * trashed during the message reception.
580  */
581 *(PACKET_HEADER *) buffer_ptr = temporary; /* COPY */
582
583 /*
584  * Send the acknowledgement to the address in conn.
585  */
586 if (send_packet(conn, conn->outgoing_buffer_base) != SUCCESS) {
587     return(FAILURE);
588 }
589
590 conn->am_timing_out = TRUE;
591 conn->time_to_live = (lm_tick * 5) + network_timeout;
592 return(PARTIAL_REQUEST);
593
594 }
595
596 }
597
598 }
599

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89	PAGE # 6/6
			TIME 4:42:06 pm	

LINE #	SOURCE TEXT
600	static u_short
601	receive_packet(conn, packet)
602	register CONNECTION *conn,
603	register char *packet,
604	{
605	int sock_size, /* sock must be made storage class register */
606	register int rcnt,
607	register int byte_count = 0,
608	register char *buffer_ptr,
609	/*
610	Initialize the receive buffer pointer and socket size.
611	*/
612	buffer_ptr = packet,
613	sock_size = SOCK_SIZE,
614	/*
615	Loop until we receive all the packet, incrementing
616	buffer_ptr and byte_count as we go.
617	*/
618	if (MODELER
619	{
620	rcnt = recvfrom(conn->fd, buffer_ptr, MAX_SERVER_PACKET, 0, (struct sockaddr *)&conn->sock, &sock_size),
621	if (rcnt <= 0) {
622	log_queue_message(SYS_ERROR_MSG, "failure to receive any bytes from the modalar"),
623	return(FAILURE),
624	}
625	byte_count += rcnt,
626	buffer_ptr += rcnt,
627	while (byte_count != BYTE_COUNT(packet)), /* Don't stop until we have all the bytes. */
628	} else
629	{
630	rcnt = recvfrom(conn->fd, buffer_ptr, MAX_SERVER_PACKET, 0, (struct sockaddr *)&conn->sock, &sock_size),
631	if (rcnt <= 0) {
632	log_queue_message(SYS_ERROR_MSG, "failure to receive any bytes from the modalar"),
633	return(FAILURE),
634	}
635	if ((SEQUENCE_NUMBER(packet) != conn->header.sequence_number) &&
636	(0 == SEQUENCE_NUMBER(packet))) {
637	reset_network_connection(conn),
638	return(FAILURE),
639	}
640	if (rcnt != BYTE_COUNT(packet)) {
641	log_queue_message(SYS_ERROR_MSG, "failure to receive expected number of bytes from the modalar(%d, %d)", rcnt, BYTE_COUNT(packet)),
642	return(FAILURE),
643	}
644	}
645	return(SUCCESS),
646	}
647	
648	void
649	tx_task()
650	{
651	struct file_entry file,
652	while(1)
653	{
654	file.file_no = TX_FIFO,
655	file.task = TRANSMIT_TASK_ID,
656	if (file_get(&file) == FAILURE)
657	{
658	vprintf("failure to get packet from file in transmit task\n"),
659	continue,
660	}
661	(void)send_packet_to_host(table_of_conns[file.user], file.data),
662	}
663	}
664	
665	static u_short
666	send_packet(conn, packet)
667	register CONNECTION *conn,
668	register char *packet,
669	{
670	/*
671	Unconditionally send the packet to the connected user. Other routines
672	determine whether the packet actually arrived at its destination.
673	*/
674	if (MODELER
675	{
676	struct file_entry file,
677	file.user = conn->fd,
678	file.data = packet,
679	file.task = NO_TASK,
680	file.file_no = TX_FIFO,
681	return(file_put(&file)),
682	} else
683	{
684	return(send_packet_to_host(conn, packet)),
685	}
686	}
687	
688	static u_short
689	send_packet_to_host(conn, packet)
690	register CONNECTION *conn,
691	register char *packet,
692	{
693	register int rcnt,
694	register int byte_count,
695	/*
696	Set the number of bytes we have to send.
697	*/
698	byte_count = BYTE_COUNT(packet),
699	/*
700	

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89	PAGE # 8/8
LINE #		SOURCE TEXT		
839		} else vprintf("LMSillive user number out of range\n");		
840		}		
841		} else if(*ptr == "dead") {		
842		proc_id = *ptr;		
843		user = *(u_char *)++ptr;		
844		if(user < MAX_USERS) {		
845		if(table_of_conns[user] != 0 && table_of_conns[user]->header.process_id == proc_id) {		
846		abort_user(user);		
847	#ifdef DIAGS	return(close_connection_for_server(table_of_conns[user]));		
848		}		
849		else		
850		in_dead_user_mask = 1 << user;		
851	#endif DIAGS	}		
852		} else vprintf("LMSidead proc ids did not match %d, %d\n", proc_id, user);		
853		}		
854		} else vprintf("LMSidead user number out of range\n");		
855		}		
856		} else if(*ptr == "LMSkill") {		
857		proc_id = *ptr;		
858		user = *(u_char *)++ptr;		
859		conn_table = table_of_conns[0];		
860		for(i = 0; i < MAX_USERS; i++) {		
861		conn = conn_table++;		
862		if((conn != 0) && (conn->header.process_id == proc_id) &&		
863		(conn->sock_address == ip_header->source_address)) {		
864		vprintf("LMSkill user %d\n", i);		
865		abort_user(user);		
866	#ifdef DIAGS	return(close_connection_for_server(table_of_conns[user]));		
867		}		
868		else		
869		in_dead_user_mask = 1 << user;		
870	#endif DIAGS	}		
871		}		
872		}		
873		}		
874		}		
875		}		
876		return(SUCCESS);		
877		}		
878		}		
879		}		
880		u_abort		
881		in_choose_connection(ret_user)		
882		u_char *ret_user;		
883		{		
884		u_char user, new_user;		
885		u_short rta;		
886		int sock_size;		
887		CONNECTION *rta_conn;		
888		{		
889		#ifdef MODELER		
890		struct fifo_entry fifo;		
891		#endif		
892		char my_buffer(P_HEADER_SIZE);		
893		{		
894		while(1) {		
895		{		
896		/* Peek at the data on the incoming socket. No data is actually read, the		
897		purpose is to obtain the source address of the next available request.		
898		*/		
899		sock_size = SOCK_SIZE;		
900		if (recvfrom(base_conn.fd, my_buffer, P_HEADER_SIZE, MSG_PEEK, (struct sockaddr *)&base_conn.sock, &sock_size) <= 0) {		
901		vprintf("recvfrom() error while checking connection\n");		
902		(void)recvfrom(base_conn.fd, my_buffer, P_HEADER_SIZE, 0, (struct sockaddr *)&base_conn.sock, &sock_size);		
903		return(Failure);		
904		}		
905		if(check_for_deadmsg(my_buffer) == SUCCESS)		
906		continue;		
907		/* Find or create a connection for the just peeked at message.		
908		*/		
909		if (find_or_create_a_connection(base_conn, rta_conn, user, &new_user) != SUCCESS) {		
910		/* Close up the new perhaps network packet peeked at above. */		
911		if (recvfrom(base_conn.fd, &my_buffer, MAX_SERVER_PACKET, 0, (struct sockaddr *)&base_conn.sock, &sock_size) <= 0) {		
912		vprintf("Failure to receive any bytes from the modeler, port COM\n");		
913		return(Failure);		
914		}		
915		if(new_user) {		
916		table_of_conns[user]->header.process_id = PROCESS_ID(my_buffer);		
917		}		
918		*ret_user = user;		
919		if(rta_conn->is_sending == TRUE) {		
920		return(PENDING);		
921		}		
922		else {		
923		rta = in_receive_request(rta_conn);		
924		if ((rta == FULL_REQUEST)		
925		((rta_conn->is_closing == TRUE)		
926		(rta_conn->is_closing == TRUE)))		
927		{		
928		return(FAILURE);		
929		}		
930		if (rta == FULL_REQUEST) {		
931		rta_conn->is_processing = TRUE;		
932		fifo.fifo_no = RX_FIFO;		
933	#ifdef MODELER	fifo.data = (char *) rta_conn;		
934		fifo.user = user;		
935		fifo.task = RECEIVE_TASK_ID;		
936	#ifdef DIAGS	diag_fifo_put(&fifo); /* for RX_FIFO */		
937		}		
938	#else DIAGS	fifo_put(&fifo);		
939	#endif DIAGS	{		
940	#endif	return(SUCCESS);		
941		}		
942		if (rta == FAILURE) {		
943		if (new_user) {		
944		(void) close_connection_for_server(rta_conn);		
945		}		
946		return(FAILURE);		
947		}		
948		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89	PAGE # 9/9
LINE #		SOURCE TEXT		
959		}		
960		}		
961		}		
962		}		
963		static u_short		
964		find_or_create_connection(base_conn, rts_conn, user, new_user)		
965		register CONNECTION *base_conn;		
966		register u_char *user, *new_user;		
967		CONNECTION **rts_conn;		
968		{		
969		register CONNECTION *conn;		
970		register CONNECTION *new_conn;		
971		register CONNECTION **conn_table;		
972		register u_char i;		
973		register u_char next_free = MAX_USERS+1;		
974		register u_char active_users_found;		
975		u_long command;		
976		static char string[] = "Exceed maximum number of users on the modeler";		
977		char *string_ptr = string;		
978		{		
979		/*new_user = 0;		
980		/* A three part process exists to find a connection struct		
981		/* to return to the higher level procedure.		
982		*/		
983		/*		
984		/* 1. If there are no active users, make the first connection		
985		/* structure and use it.		
986		*/		
987		if (active_users == 0) {		
988		/*new_user = 1;		
989		if ((conn=(CONNECTION *) malloc(sizeof(CONNECTION))) == (CONNECTION *) NULL) {		
990		/* queue message(SYS_ERROR_MSG, "out of memory on modeler for network connections");		
991		return(FAILURE);		
992		}		
993		/* Initialize the new connection. */		
994		if (init_connection_for_server(conn) != SUCCESS) {		
995		/*free((char *)conn);		
996		return(FAILURE);		
997		}		
998		/* Set up the proper connection values. */		
999		conn->id = base_conn->id;		
1000		conn->sock.family = base_conn->sock.family;		
1001		conn->sock.port = base_conn->sock.port;		
1002		conn->sock.address = base_conn->sock.address;		
1003		conn->sock.packet_type = base_conn->sock.packet_type;		
1004		conn->sock.destination[0] = base_conn->sock.destination[0];		
1005		conn->sock.destination[1] = base_conn->sock.destination[1];		
1006		conn->sock.destination[2] = base_conn->sock.destination[2];		
1007		conn->sock.destination[3] = base_conn->sock.destination[3];		
1008		conn->sock.destination[4] = base_conn->sock.destination[4];		
1009		conn->sock.destination[5] = base_conn->sock.destination[5];		
1010		conn->header.process_id = base_conn->header.process_id;		
1011		{		
1012		/*user = 0;		
1013		#ifdef MODELER		
1014		conn->id = *user;		
1015		#endif		
1016		table_of_conns [*user] = conn;		
1017		/*active users;		
1018		*rts_conn = conn;		
1019		return(SUCCESS);		
1020		}		
1021		/*		
1022		/* 2. Must already have active users. Look through the current		
1023		/* list of connections for a matching port and address.		
1024		*/		
1025		conn_table = table_of_conns [0];		
1026		active_users_found = active_users;		
1027		for(i = 0; i < MAX_USERS; i++) {		
1028		/*conn = *conn_table++;		
1029		if(conn != (CONNECTION *) NULL) {		
1030		/*active_users_found;		
1031		if ((conn->sock.port == base_conn->sock.port) &&		
1032		(conn->sock.address == base_conn->sock.address))		
1033		{		
1034		/*user = i;		
1035		#ifdef MODELER		
1036		conn->id = *user;		
1037		conn->on_time_out = FALSE;		
1038		conn->number_of_live_retries = 0;		
1039		#endif		
1040		*rts_conn = conn;		
1041		return(SUCCESS);		
1042		}		
1043		/*		
1044		/* else { if(next_free == MAX_USERS+1)		
1045		/*next_free = 1;		
1046		/*		
1047		/* we found all the active users there are		
1048		/* make sure we know next free location		
1049		*/		
1050		if (active_users_found == 0) {		
1051		/*if(next_free == MAX_USERS+1)		
1052		/*next_free = 1+1;		
1053		/*break;		
1054		/*		
1055		/* we have max users already		
1056		*/		
1057		if (active_users == MAX_USERS) {		
1058		/* queue message(ERROR_MSG, "exceeded maximum number of users. id, on the modeler", MAX_USERS);		
1059		}		

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89 TIME 4:42:06 pm	PAGE # 10/10
LINE #	SOURCE TEXT		
1079	if(set_up_out_of_luck_user(base_conn) == FAILURE)		
1080	return(FAILURE);		
1081	new_conn = table_of_conns(MAX_USERS);		
1082	/*		
1083	get the packet		
1084	*/		
1085	if (receive_packet(new_conn, new_conn->incoming_buffer_base) != SUCCESS) {		
1086	(void) close_connection_for_server(new_conn);		
1087	return(FAILURE);		
1088	}		
1089	/*		
1090	get a message to transmit		
1091	*/		
1092	IN_RESET_INCOMING(new_conn);		
1093	IN_RESET_OUTGOING(new_conn);		
1094	command = IN_GET_LONG(new_conn);		
1095	command++;		
1096	IN_PUT_LONG(new_conn, command);		
1097	IN_PUT_LONG(new_conn, 1);		
1098	IN_PUT_CHAR(new_conn, ERROR_MSG);		
1099	while("string_ptr")		
1100	{		
1101	IN_PUT_CHAR(new_conn, "string_ptr++);		
1102	}		
1103	IN_PUT_CHAR(new_conn, 0);		
1104	/*		
1105	get msg to transmit is 100		
1106	*/		
1107	(void) in_send_reply(new_conn);		
1108	/*		
1109	set close connection for server		
1110	*/		
1111	(void) set_close_connection_for_server(new_conn);		
1112	return(FAILURE);		
1113	}		
1114	/*		
1115	J. Next we have a brand new connection. Create a new connection		
1116	structure, initialize that structure, add the new connection		
1117	into the connection table, and use it.		
1118	*/		
1119	new_user = 1;		
1120	if ((new_conn = (CONNECTION *) malloc(sizeof(CONNECTION))) == (CONNECTION *) NULL) {		
1121	in_send_message(STATUS_ERROR_MSG, "out of memory on modeler for network connections");		
1122	return(FAILURE);		
1123	}		
1124	/*		
1125	Initialize the new connection		
1126	*/		
1127	if (init_connection_for_server(new_conn) != SUCCESS) {		
1128	free((char *)new_conn);		
1129	return(FAILURE);		
1130	}		
1131	/*		
1132	Set up the proper connection values		
1133	*/		
1134	new_conn->id = base_conn->id;		
1135	new_conn->sock.family = base_conn->sock.family;		
1136	new_conn->sock.port = base_conn->sock.port;		
1137	new_conn->sock.address = base_conn->sock.address;		
1138	new_conn->sock.packet_type = base_conn->sock.packet_type;		
1139	new_conn->sock.destination[0] = base_conn->sock.destination[0];		
1140	new_conn->sock.destination[1] = base_conn->sock.destination[1];		
1141	new_conn->sock.destination[2] = base_conn->sock.destination[2];		
1142	new_conn->sock.destination[3] = base_conn->sock.destination[3];		
1143	new_conn->sock.destination[4] = base_conn->sock.destination[4];		
1144	new_conn->sock.destination[5] = base_conn->sock.destination[5];		
1145	new_conn->header.process_id = base_conn->header.process_id;		
1146	/*		
1147	user = next_free,		
1148	*/		
1149	#ifdef MODELER		
1150	new_conn->id = "user";		
1151	#endif		
1152	table_of_conns["user"] = new_conn;		
1153	/*		
1154	active users,		
1155	*/		
1156	return(SUCCESS);		
1157	}		
1158	/*		
1159	close connection for server(conn)		
1160	*/		
1161	register CONNECTION *conn;		
1162	{		
1163	/*		
1164	WARNING: If you change this routine check reset_network_connection()		
1165	to see if the conn changes are needed there.		
1166	*/		
1167	#ifdef MODELER		
1168	register u_char i;		
1169	register CONNECTION *next_conn;		
1170	register CONNECTION *conn_table;		
1171	#endif		
1172	/*		
1173	The users > MAX_USERS are not real users.		
1174	They are used to send messages back to the host.		
1175	*/		
1176	if(conn->id < MAX_USERS)		
1177	{		
1178	--active_users;		
1179	(void) free(conn->incoming_buffer_base);		
1180	(void) free(conn->outgoing_buffer_base);		
1181	}		
1182	#ifdef MODELER		
1183	/*		
1184	On the modeler the file descriptor is the index into the connection		
1185	table, therefore, we don't need to look through the table to find		
1186	what needs to be closed/cleaned-up.		
1187	*/		
1188	#endif		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89	PAGE # 11/11
LINE #		SOURCE TEXT		
1199		if (close(conn->fd) != 0) {		
1200		log_queue_message(SYS_ERROR_MSG, "failure to close modeler communication endpoint(socket)");		
1201		return(FAILURE);		
1202		}		
1203		return(SUCCESS);		
1204		}		
1205		/*		
1206		* search thru table of connections and delete appropriate		
1207		* connection.		
1208		*/		
1209		conn_table = table_of_conns[0];		
1210		for(i = 0, i < MAX_USERS; ++i) {		
1211		if(conn_table[i] == (CONNECTION *) NULL) {		
1212		if (conn->sock.port == conn_table[i]->sock.port) &&		
1213		(conn->sock.address == conn_table[i]->sock.address) {		
1214		(void) free((char *)table_of_conns[i]);		
1215		table_of_conns[i] = (CONNECTION *) NULL;		
1216		return(SUCCESS);		
1217		}		
1218		}		
1219		}		
1220		}		
1221		printf("could not find user to close\n");		
1222		return(FAILURE);		
1223		}		
1224		#endif		
1225		static u_short		
1226		init_connection_for_server(conn)		
1227		register CONNECTION *conn;		
1228		{		
1229		/*		
1230		* WARNING: if you change this routine check reset_network_connection()		
1231		* to see if the conn changes are needed there.		
1232		*/		
1233		/*		
1234		* set default values into the packet header construct that is		
1235		* used to track the state of the host communication.		
1236		*/		
1237		conn->header.byte_count = -1;		
1238		conn->header.process_id = -1;		
1239		conn->header.sequence_number = 1;		
1240		conn->header.partial_packet_count = 0;		
1241		/*		
1242		* Initialize the partial packet indicators. (Technically, these don't		
1243		* need to be initialized for the client as the client never references		
1244		* them, but what the heck -- let's be consistent and init everywhere.)		
1245		*/		
1246		conn->am_sending = FALSE;		
1247		conn->bytes_left_to_send = 0;		
1248		conn->am_receiving = FALSE;		
1249		conn->bytes_left_to_receive = 0;		
1250		conn->am_closing = FALSE;		
1251		conn->do_close = FALSE;		
1252		conn->am_timing_out = FALSE;		
1253		conn->time_at_last_response = ln_tick;		
1254		conn->time_to_live = 0;		
1255		conn->am_processing = FALSE;		
1256		/*		
1257		* Set the incoming buffer time, and of current message,		
1258		* buffer size, and allocate memory space to receive messages.		
1259		*/		
1260		conn->incoming_time = 0;		
1261		conn->incoming_buffer_size = 2*MAX_SERVER_PACKET;		
1262		if ((conn->incoming_buffer_base = malloc(2*MAX_SERVER_PACKET)) == (char *) NULL) {		
1263		log_queue_message(SYS_ERROR_MSG, "out of memory on modeler for incoming messages");		
1264		return(FAILURE);		
1265		}		
1266		conn->incoming_message_and = conn->incoming_buffer_base;		
1267		/*		
1268		* Initialize the incoming buffer pointer properly.		
1269		*/		
1270		LN_RESET_INCOMING(conn);		
1271		/*		
1272		* Set the outgoing buffer time, and of buffer,		
1273		* buffer size, and allocate memory space to send messages.		
1274		*/		
1275		conn->outgoing_time = 0;		
1276		conn->outgoing_buffer_size = MAX_SERVER_PACKET;		
1277		if ((conn->outgoing_buffer_base = malloc(MAX_SERVER_PACKET)) == (char *) NULL) {		
1278		free((char *)conn->incoming_buffer_base);		
1279		log_queue_message(SYS_ERROR_MSG, "out of memory on modeler for outgoing messages");		
1280		return(FAILURE);		
1281		}		
1282		conn->outgoing_buffer_and = conn->outgoing_buffer_base + conn->outgoing_buffer_size;		
1283		/*		
1284		* Initialize the outgoing buffer pointer properly.		
1285		*/		
1286		LN_RESET_OUTGOING(conn);		
1287		return(SUCCESS);		
1288		}		
1289		}		
1290		reset_network_connection(conn)		
1291		CONNECTION *conn;		
1292		{		
1293		/*		
1294		* This routine completely resets the conn structure.		
1295		* Note that most of the code has been plagiarized from		
1296		* close_conn_for_server() and init_conn_for_server()		
1297		* If either of these routines changes, check to see that		
1298		* this code does not need to be changed.		
1299		* These routines were not called directly for 2 reasons:		
1300		* 1) The user number could change, and 2) if there are		
1301		* the maximum number of users on the modeler the init		
1302		* could fail and the connection would be lost.		
1303		*/		
1304		}		
1305		}		
1306		}		
1307		}		
1308		}		
1309		}		
1310		}		
1311		}		
1312		}		
1313		}		
1314		}		
1315		}		
1316		}		
1317		}		
1318		}		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89 TIME 4:42:06 pm	PAGE # 12/12
SOURCE TEXT				
LINE #				
319	(void) free(coas->incoming_buffer_base);			
320	(void) free(coas->outgoing_buffer_base);			
321				
322	/*			
323	* Set default values into the packet header construct that is			
324	* used to track the state of the host communication.			
325	*/			
326	coas->header.byte_count = -1;			
327	coas->header.sequence_number = 1;			
328	coas->header.partial_packet_count = 0;			
329				
330	/*			
331	* Initialize the partial packet indicators. (Technically, these don't			
332	* need to be initialized for the client as the client never references			
333	* them, but what the heck - let's be consistent and init everywhere.)			
334	*/			
335	coas->as_sending = FALSE;			
336	coas->bytes_left_to_send = 0;			
337	coas->as_receiving = FALSE;			
338	coas->bytes_left_to_receive = 0;			
339	coas->as_timing_out = TRUE;			
340	coas->time_at_last_response = lm_tick;			
341	coas->time_to_live = (lm_tick * 5) + network_timeout;			
342	coas->as_processing = FALSE;			
343				
344	/*			
345	* Set the incoming buffer time, end of current message,			
346	* buffer size, and allocate memory space to receive messages.			
347	*/			
348	coas->incoming_time = 0;			
349	coas->incoming_buffer_size = 2*MAX_SERVER_PACKET;			
350	if ((coas->incoming_buffer_base = malloc(2*MAX_SERVER_PACKET)) == (char *) NULL) {			
351	in_queue_message(SYS_ERROR_MSG, "out of memory on modular for incoming messages");			
352	return(FAILURE);			
353	}			
354	coas->incoming_message_end = coas->incoming_buffer_base;			
355				
356	/*			
357	* Initialize the incoming buffer pointer properly.			
358	*/			
359	LM_RESET_INCOMING(coas);			
360				
361	/*			
362	* Set the outgoing buffer time, end of buffer,			
363	* buffer size, and allocate memory space to send messages.			
364	*/			
365	coas->outgoing_time = 0;			
366	coas->outgoing_buffer_size = MAX_SERVER_PACKET;			
367	if ((coas->outgoing_buffer_base = malloc(MAX_SERVER_PACKET)) == (char *) NULL) {			
368	free((char *) coas->incoming_buffer_base);			
369	in_queue_message(SYS_ERROR_MSG, "out of memory on modular for outgoing messages");			
370	return(FAILURE);			
371	}			
372	coas->outgoing_buffer_end = coas->outgoing_buffer_base + coas->outgoing_buffer_size;			
373				
374	/*			
375	* Initialize the outgoing buffer pointer properly.			
376	*/			
377	LM_RESET_OUTGOING(coas);			
378	return(SUCCESS);			
379				
380				
381				
382	#ifdef MODULAR			
383	{			
384	close(fd);			
385	int id;			
386	{			
387	if (table_of_coas[id] != (CONNECTION *) NULL) {			
388	(void) free((char *) table_of_coas[id]);			
389	table_of_coas[id] = (CONNECTION *) NULL;			
390	return(0);			
391	}			
392	return(-1);			
393	}			
394				
395	int			
396	socket(family, type, protocol)			
397	int family,			
398	int type,			
399	int protocol,			
400	{			
401	/*			
402	* The modular does not use the file descriptor that the UNIX socket(2)			
403	* call returns. Just return "1", which means success by the UNIX definition.			
404	*/			
405	return(1);			
406	}			
407				
408	int			
409	bind(fd, sock, size)			
410	int fd,			
411	struct sockaddr *sock,			
412	int size,			
413	{			
414	/*			
415	* The modular does not use any thing that the UNIX bind(2) call returns			
416	* or sets. Just return "0", which means success by the UNIX definition.			
417	*/			
418	return(0);			
419	}			
420	#endif			
421				
422	u_short			
423	set_close_connection_for_server(coas)			
424	register CONNECTION *coas,			

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM networkm/net_srv.c	DATE 5/23/89 TIME 4:42:06 pm	PAGE # 13/13
LINE #	SOURCE TEXT		
1439	{		
1440	if(conn -> am_closing == TRUE)		
1441	return(FAILURE);		
1442	conn -> am_closing = TRUE;		
1443	return(SUCCESS);		
1444	}		
1445	u_short		
1446	check_connection_for_life(conn)		
1447	CONNECTION *conn;		
1448	{		
1449	SOCKET_ADDRESS sock;		
1450	int scnt;		
1451	{		
1452	LM_RESET_RAW_TX_BUFFER((char) conn->id);		
1453	LM_PUT_CHAR_TX_BUFFER('1');		
1454	LM_PUT_CHAR_TX_BUFFER('M');		
1455	LM_PUT_CHAR_TX_BUFFER('S');		
1456	LM_PUT_CHAR_TX_BUFFER('I');		
1457	LM_PUT_CHAR_TX_BUFFER('X');		
1458	LM_PUT_CHAR_TX_BUFFER('I');		
1459	LM_PUT_CHAR_TX_BUFFER('S');		
1460	LM_PUT_CHAR_TX_BUFFER('O');		
1461	LM_PUT_CHAR_TX_BUFFER('C');		
1462	LM_PUT_LONG_TX_BUFFER(conn -> header.process_id);		
1463	LM_PUT_CHAR_TX_BUFFER((char) conn->id);		
1464	}		
1465	sock = conn->sock;		
1466	sock.port=host_internet_socket_number;		
1467	scnt = sendto(0, tx_buffer(conn->id), LM_TX_BYTECOUNT((char) conn->id),UDP_IP_PACKET, (struct sockaddr *)sock, sizeof(SOCKET_ADDRESS		
1468);		
1469	if(scnt < 12)		
1470	{		
1471	fprintf("failure to transmit LMSiproc packet to host\n");		
1472	return(FAILURE);		
1473	}		
1474	return(SUCCESS);		
1475	}		
1476	#endif		
1477	#ifdef DIAGS		
1478	short_user(user)		
1479	{		
1480	/* do not abort if running diag		
1481	*/		
1482	return;		
1483	}		
1484	else		
1485	#include "device.h"		
1486	#include "hardware.h"		
1487	#include "server.h"		
1488	#include "lmserv.h"		
1489	short_user_number(userno)		
1490	u_char usernr;		
1491	{		
1492	short_user(user_info_array[userno]);		
1493	}		
1494	#endif		
1495	static u_short		
1496	set_up_out_of_lack_user(base_conn)		
1497	CONNECTION *base_conn,		
1498	{		
1499	register CONNECTION *new_conn;		
1500	{		
1501	/*		
1502	/* is the extra user currently in use		
1503	*/		
1504	if(table_of_conns[MAX_USERS] != (CONNECTION *) NULL)		
1505	{		
1506	return(FAILURE);		
1507	}		
1508	/*		
1509	/* Initialize the new connection.		
1510	*/		
1511	if ((new_conn = (CONNECTION *) malloc(sizeof(CONNECTION))) == (CONNECTION *) NULL) {		
1512	lm_queue_message(STS_ERROR_MSG, "out of memory on modular for network connections");		
1513	return(FAILURE);		
1514	}		
1515	/*		
1516	/* Initialize the new connection.		
1517	*/		
1518	if (init_connection_for_server(new_conn) != SUCCESS) {		
1519	return(FAILURE);		
1520	}		
1521	/*		
1522	/* Set up the proper connection values.		
1523	*/		
1524	new_conn->id = (int)MAX_USERS;		
1525	new_conn->sock.family = base_conn->sock.family;		
1526	new_conn->sock.port = base_conn->sock.port;		
1527	new_conn->sock.address = base_conn->sock.address;		
1528	new_conn->sock.packet_type = base_conn->sock.packet_type;		
1529	new_conn->sock.destination[0] = base_conn->sock.destination[0];		
1530	new_conn->sock.destination[1] = base_conn->sock.destination[1];		
1531	new_conn->sock.destination[2] = base_conn->sock.destination[2];		
1532	new_conn->sock.destination[3] = base_conn->sock.destination[3];		
1533	new_conn->sock.destination[4] = base_conn->sock.destination[4];		
1534	new_conn->sock.destination[5] = base_conn->sock.destination[5];		
1535	new_conn->sock.destination[6] = base_conn->sock.destination[6];		
1536	new_conn->header.process_id = base_conn->header.process_id;		
1537	/*		
1538	/* if more than 1, extra user this needs to be changed.		
1539	*/		
1540	table_of_conns[MAX_USERS] = new_conn;		
1541	return(SUCCESS);		
1542	}		
1543	}		
1544	{		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM	DATE	PAGE #
		networkc/net_com.c	5/23/89	
			TIME	1/1
			1:20:44 pm	
SOURCE TEXT				
LINE #				
1	/* SCCS ID: net_com.c Rev 3.1.1 4/24/89 at 08:02:38 */			
2	#include "common.h"			
3	#include "message.h"			
4	#include "network.h"			
5				
6	void			
7	in_extend_outgoing_buffer(conn)			
8	{			
9	CONNECTION *conn;			
10	{			
11	register char *ptr;			
12	register u_long new_buffer_size;			
13	register u_long previous_pointer_offset;			
14	{			
15	/*			
16	* Save the offset of the current buffer pointer so that it can be restored			
17	* in the (likely) case that realloc() moves the buffer.			
18	*/			
19	previous_pointer_offset = (u_long)conn->outgoing_buffer_pointer - (u_long)conn->outgoing_buffer_base;			
20	/*			
21	* Calculate the size of the new buffer.			
22	*/			
23	new_buffer_size = conn->outgoing_buffer_size + OUTGOING_BUFFER_INCREMENT;			
24	/*			
25	* Increase the buffer size as calculated above.			
26	*/			
27	ptr = realloc(conn->outgoing_buffer_base, (unsigned)new_buffer_size);			
28	if (ptr == (char *)NULL) {			
29	in_queue_message(SYS_ERROR_MSG, "error, could not (re)allocate space for outgoing nodular messages");			
30	}			
31	else {			
32	/*			
33	* Update all the required pointers and counts.			
34	*/			
35	conn->outgoing_buffer_base = ptr;			
36	conn->outgoing_buffer_pointer = ptr + previous_pointer_offset;			
37	/*			
38	* Calculate the size of the new buffer.			
39	*/			
40	conn->outgoing_buffer_size = new_buffer_size;			
41	conn->outgoing_buffer_end = conn->outgoing_buffer_base + conn->outgoing_buffer_size;			
42	}			
43	}			
44				
45				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/hprescan.c	DATE 5/23/89 TIME 1:21:08 pm	PAGE # 1/1
SOURCE TEXT				
LINE #				
1	/* SCCS_ID: Hprescan.c rev 1.1, 4/24/89 at 08:00:06 */			
2	/*			
3	/* preprocessor for HML			
4	*/			
5	/*			
6	/* include "HML.h"			
7	/* include "common.h"			
8	/* include <stdio.h>			
9	*/			
10	/*			
11	/* define BUF_SIZE 1024			
12	/* define BUF_INCR 1024			
13	/* define MAXFILENAMELEN 128			
14	/* define MAXINCLUDEFILENAMELEN 256			
15	/* define LM_LIBRARY_VARIABLE "LM_LIB"			
16	/*			
17	extern char *strcpy () ;			
18	static short			
19	nextc /* preprocessor's lookahead character */			
20	oldnextc /* preprocessor's lookahead character save space */			
21	static char			
22	*output /* pointer into output buffer */			
23	*source_name /* source file name */			
24	*old_name /* to save source name */			
25	*source_text /* pointer to source text */			
26	directive [31] /* name of directive */			
27	directive2 [32] /* another name of directive */			
28	filename [MAXFILENAMELEN] /* file name buffer */			
29	includefilename [MAXINCLUDEFILENAMELEN] ;			
30	/* include file name buffer */			
31	static unsigned long			
32	mark /* to mark start of token */			
33	out_index /* index into output buffer */			
34	out_length /* length of output buffer */			
35	static ushort			
36	malloc_failed /* to indicate malloc() failure */			
37	need_filename /* indicates that a filename must be next */			
38	lineno /* current line number */			
39	oldlineno /* to save line number */			
40	newlineno /* to save new line number */			
41	newlineno /* to delay line number incrementing */			
42	including /* flag indicating that we are including */			
43	ccount /* comment nesting count */			
44	static FILE *include_file /* pointer to include file descriptor */			
45	/*			
46	static void			
47	pregetnext ()			
48	/* gets the next character for the pre-scanner */			
49	{ if (including)			
50	nextc = getc (include_file) ;			
51	else			
52	{ nextc = *source_text++ ;			
53	if (nextc == '\0')			
54	nextc = EOF ;			
55	}			
56	if (nextc == '\n')			
57	++newlineno ;			
58	} /* pregetnext */			
59	/*			
60	void			
61	lm_init_prescan_info ()			
62	{ lm_init_lineno (lineno) ;			
63	lm_init_input (source_name) ;			
64	} /* lm_init_prescan_info */			
65	/*			
66	/*VARARGS1*/			
67	static void			
68	lerror (format , arg , arg2)			
69	char			
70	*format ,			
71	*arg ,			
72	*arg2 ;			
73	/* reports other lexical errors via lm_error() */			
74	{ lm_init_prescan_info () ;			
75	lm_error (1 , format , arg , arg2) ;			
76	} /* lerror */			
77	/*			
78	static void			
79	lm_lexical_error ()			
80	/* reports a lexical error via lm_error() */			
81	{ char tmp [2] ;			
82	*tmp = nextc ;			
83	tmp [1] = '\0' ;			
84	lm_init_lineno (newlineno) ;			
85	lm_init_input (source_name) ;			
86	lm_error			
87	(1 ,			
88	/*MSG*/"lexical error: character is illegal"			
89	, tmp			
90) ;			
91	} /* lm_lexical_error */			
92	/*			
93	static ushort			
94	is_directive (str)			
95	char *str ;			
96	/* returns 1 iff the passed string is an include directive */			
97	{ if (strcmp (str , "physical") == 0)			
98	return 1 ;			
99	if (strcmp (str , "timing") == 0)			
100	return 1 ;			
101	if (strcmp (str , "package") == 0)			
102	return 1 ;			
103	if (strcmp (str , "adapter") == 0)			
104	return 1 ;			
105	if (strcmp (str , "names") == 0)			
106	return 1 ;			
107	if (strcmp (str , "options") == 0)			
108	return 1 ;			
109	return 0 ;			
110	} /* is_directive */			
111	/*			
112	static void			
113	get_unquoted_filename ()			
114	/* generates into filename the current token, which is unquoted */			
115	{ char			
116	*str ,			
117	*in ,			
118	str = output + out_index ,			
119	in = filename ,			
120	do			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/h/Prescan.c	DATE 5/23/89	PAGE # 2/2
LINE #		SOURCE TEXT		
121		/*f++ = *str+ */ /* need to make sure it fits */		
122		while (*str != '\0')		
123		*fa = '\0' ;		
124		/* get_unquoted_filename */		
125		static void		
126		get_single_quoted_filename ()		
127		/* generates into filename the current token, which is single quoted */		
128		{ char		
129		*str ,		
130		*fa ,		
131		str = output + out_index ,		
132		++str ,		
133		fa = filename ,		
134		do		
135		{ if (*str == '\\')		
136		++str ;		
137		*f++ = *str , /* need to make sure it fits */		
138		if (*str != '\0')		
139		++str ;		
140		}		
141		while (*str != '\0')		
142		/* get_single_quoted_filename */		
143		static void		
144		get_double_quoted_filename ()		
145		/* generates into filename the current token, which is double quoted */		
146		{ char		
147		*str ,		
148		*fa ,		
149		str = output + out_index ,		
150		++str ,		
151		fa = filename ,		
152		do		
153		{ if (*str == '"')		
154		++str ;		
155		*f++ = *str , /* need to make sure it fits */		
156		if (*str != '\0')		
157		++str ;		
158		}		
159		while (*str != '\0')		
160		/* get_double_quoted_filename */		
161		static void		
162		out_char (c)		
163		/* outputs character to output buffer, extending if necessary */		
164		{ static char last_c = ' ' ;		
165		if (malloc_failed)		
166		return ;		
167		if (last_c != ' ' && c != ' ' && c != '\n')		
168		return ;		
169		if (out_index == out_length - 2)		
170		{ output = realloc (output , (unsigned) (out_length + BUF_INCR)) ;		
171		if (output == NULL)		
172		{ lm_Error		
173		{ 1 ,		
174		/*MSG*/"out of memory on host (out_char)"		
175		return ;		
176		malloc_failed = 1 ;		
177		return ;		
178		}		
179		output [out_index++] = c ;		
180		last_c = c ;		
181		/* out_char */		
182		static void		
183		out_control (line_number , file_name)		
184		/* outputs a line control line via out_char */		
185		{ char		
186		*ptr		
187		sum [12] ;		
188		out_char ('0') ;		
189		out_char (' ') ;		
190		{ void) sprintf (sum , "%u", line_number) ;		
191		for (ptr = sum , *ptr != '\0' ; ++ptr)		
192		out_char (*ptr) ;		
193		out_char (' ') ;		
194		for (, *file_name != '\0' ; ++file_name)		
195		out_char (*file_name) ;		
196		out_char ('\n') ;		
197		/* out_control */		
198		void		
199		lm_init_prescan (name , text)		
200		{ char		
201		*name ,		
202		*text ,		
203		/* initializes the prescanner */		
204		{ output = malloc ((unsigned) (out_length + BUF_SIZE)) ;		
205		if (output == NULL)		
206		{ lm_Error		
207		{ 1 ,		
208		/*MSG*/"out of memory on host (lm_init_prescan)"		
209		return ;		
210		malloc_failed = 1 ;		
211		else		
212		{ *output = '\0' ;		
213		malloc_failed = 0 ;		
214		}		
215		out_index = 0 ;		
216		including = 0 ;		
217		lineno = 1 ;		
218		maxlineno = 1 ;		
219		read_filename = 0 ;		
220		source_name = name ;		
221		source_text = text ;		
222		out_control (lineno , source_name) ;		
223		lm_init_input (name) ;		
224		lm_init_lineno (lineno) ;		
225		prescanner () ;		
226		/* lm_init_prescan */		
227		char *		
228		lm_prescan ()		
229		/* prescan a token from source_text, output it to output buffer */		
230		{		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/Hprescan.c	DATE 5/23/89	PAGE # 3/3
LINE #		SOURCE TEXT		
241		{ while (lineno = newlineno , sentc != EOF including)		
242		{ if (sentc == EOF)		
243		{ (void) fseek (include_file ,		
244		including = 0 ;		
245		sentc = oldsentc ;		
246		newlineno = oldnewlineno ;		
247		lineno = oldlineno ;		
248		source_name = old_name ;		
249		out_char ("\n") ;		
250		out_control (lineno , source_name) ;		
251		}		
252		else if (sentc < ' ')		
253		{ if (need_filename == 0)		
254		out_char (sentc) ;		
255		pregment () ;		
256		}		
257		else if		
258		{ sentc >= 'A' && sentc <= 'Z'		
259		sentc >= 'a' && sentc <= 'z'		
260		sentc == '.' }		
261		{ mark = out_index ;		
262		out_char (sentc) ;		
263		while		
264		{ (pregment () , sentc) >= 'A' && sentc <= 'Z'		
265		sentc >= 'a' && sentc <= 'z'		
266		sentc >= '0' && sentc <= '9'		
267		sentc == '.' }		
268		sentc == '\n' }		
269		sentc == '\t' }		
270		}		
271		out_char (sentc) ;		
272		output [out_index] = '\0' ;		
273		if (need_filename)		
274		{ out_index = mark ;		
275		get_unquoted_filename () ;		
276		goto get_in ;		
277		}		
278		if (is_directive (output + mark))		
279		{ (void) strcpy (directive2 , output + mark) ;		
280		if (including)		
281		{ lexerr		
282		{		
283		/*LMSC*/"cannot reference %s files from within %s files"		
284		directive ,		
285		directive		
286		}		
287		sentc = EOF ;		
288		continue ;		
289		}		
290		{ void) strcpy (directive , directive2) ;		
291		out_index = mark ;		
292		need_filename = 1 ;		
293		continue ;		
294		}		
295		return output + mark ;		
296		}		
297		else if (sentc == '\n')		
298		{ mark = out_index ;		
299		do		
300		{ out_char ('\n') ;		
301		while		
302		{ (pregment () , sentc) != EOF &&		
303		sentc != '\n' && sentc != '\t' }		
304		{		
305		out_char (sentc) ;		
306		if (sentc == EOF)		
307		lexerr		
308		{		
309		/*LMSC*/"unterminated string: end of file encountered within a single-quoted string"		
310		}		
311		else if (sentc == '\n')		
312		lexerr		
313		{		
314		/*LMSC*/"mismatched quotes: newline encountered within a single-quoted string"		
315		}		
316		else		
317		{ out_char ('\n') ;		
318		pregment () ;		
319		}		
320		}		
321		while (sentc == '\n')		
322		output [out_index] = '\0' ;		
323		if (need_filename)		
324		{ out_index = mark ;		
325		get_single_quoted_filename () ;		
326		get_in :		
327		/*save need_filename*/		
328		need_filename = 0 ;		
329		if		
330		{ is resolve library file		
331		{ IN_LIBRARY_VARIABLE ,		
332		filename ,		
333		includefilename ,		
334		sizeof (includefilename)		
335		}		
336		{		
337		lexerr		
338		{		
339		/*LMSC*/"can't resolve %s file %s"		
340		directive ,		
341		filename		
342		}		
343		sentc = EOF ;		
344		continue ;		
345		}		
346		include_file = fopen (includefilename , "r") ;		
347		if (include_file == NULL)		
348		{ lexerr		
349		{		
350		/*LMSC*/"can't open %s file %s"		
351		directive ,		
352		includefilename		
353		}		
354		sentc = EOF ;		
355		continue ;		
356		}		
357		including = 1 ;		
358		oldsentc = sentc ;		
359		oldlineno = lineno ;		
360		oldnewlineno = newlineno ;		

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM shellsw/Hprescan.c	DATE 5/23/89 TIME 1:21:08 pm PAGE # 4/4
--	--------------------------------------	---

LINE #
SOURCE TEXT

```

361      add_name = source_name ;
362      source_name = filename ;
363      lineno = lineno + 1 ;
364      out_char ( ' ' , lineno , source_name ) ,
365      parenthesis ( ' ' ,
366      continue ;
367    }
368    return output + mark ;
369  }
370  else if ( ! mark )
371  {
372    mark = out_index ;
373    do
374    {
375      out_char ( ' ' ,
376      while
377      {
378        ( ! parenthesis ( ' ' , mark ) ) != EOF &&
379        mark != '\n' && mark != ' '
380      }
381      out_char ( mark ) ,
382      if ( mark == EOF )
383        break ;
384    }
385    /*LMSG*/"unterminated string: end of file encountered within a double-quoted string"
386    ,
387    else if ( mark == '\n' )
388      break ;
389    /*LMSG*/"unmatched quotes: newline encountered within a double-quoted string"
390    ,
391    else
392    {
393      out_char ( ' ' ,
394      parenthesis ( ' ' ,
395      while ( mark == ' ' )
396      {
397        output [ out_index ] = ' ' ;
398        if ( ! mark_filename )
399        {
400          out_index = mark ;
401          get_double_quoted_filename ( ' ' ,
402          getc get_fa ;
403        }
404        return output + mark ;
405      }
406      else if ( mark >= '0' && mark <= '9' || mark == '.' )
407      {
408        mark = out_index ;
409        out_char ( mark ) ,
410        while
411        {
412          parenthesis ( ' ' , mark ) >= 'A' && mark <= 'Z' ||
413          mark >= 'a' && mark <= 'z' ||
414          mark >= '0' && mark <= '9' ||
415          mark == '.' ||
416          mark == '-'
417        }
418        out_char ( mark ) ,
419        output [ out_index ] = ' ' ;
420        if ( ! mark_filename )
421        {
422          out_index = mark ;
423          get_double_quoted_filename ( ' ' ,
424          getc get_fa ;
425        }
426        return output + mark ;
427      }
428      else if ( mark == '[' || mark == '/' )
429      {
430        count = 0 ;
431        out_char ( ' ' ,
432        while
433        {
434          ( ! parenthesis ( ' ' , mark ) ) != '[' || count &&
435          mark != EOF
436        }
437        {
438          if ( mark == '[' )
439            count++ ;
440          else if ( mark == '[' || mark == '/' )
441            count-- ;
442          if ( mark == '\n' )
443            out_char ( mark ) ,
444          if ( mark == '[' || mark == '/' )
445            parenthesis ( ' ' ,
446          else
447            break ;
448        }
449        /*LMSG*/"unterminated comment: end of file encountered within a comment"
450        ,
451        else
452        {
453          if ( ! mark_filename )
454          {
455            break ;
456            /*LMSG*/"filename expected after %s directive"
457            , directive
458          }
459          mark = EOF ;
460          continue ;
461        }
462        mark = out_index ;
463        switch ( mark )
464        {
465          case ' ' :
466          case '!' :
467          case '%' :
468          case '-' :
469          case '=' :
470          case '[' :
471          case ']' :
472          case '~' :
473          case '^' :
474          break ;
475          default :
476            in_klitical_error ( ' ' ,
477            parenthesis ( ' ' ,
478            continue ;
479          }
480          out_char ( mark ) ,
481          parenthesis ( ' ' ,
482          output [ out_index ] = ' ' ;
483          return output + mark ;
484        }
485      }
486      return NULL ;
487    } /* in_Hprescan */
488  }
489  char *
490  in_Hdose_process ( ' ' ,
491  in_Hdose_process ( final output buffer generated by process() */

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/Hprescan.c	DATE 5/23/89	PAGE # 5/5
LINE #		SOURCE TEXT		
481		{ if (malloc_failed)		
482		return NULL ;		
483		output [out_index] = '\0' ;		
484		return output ;		
485		} /* in _sdoss_prescan.c */		
486				
487		/* end of prescaner for HML */		
488				

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/Pr_host.c	DATE 5/23/89	PAGE # 1/6
LINE #		SOURCE TEXT		
1		/* SCS ID: Pr_host.c rev 1.1, 4/24/89 at 08:00:14 */		
2		/* Processor: Sun H801, HOST AIXM		
3		*/		
4		#include <stdio.h>		
5		#include "common.h"		
6		#include "swl.h"		
7		#include "string.h"		
8		#include "sparse.h"		
9		#include "tree.h"		
10		#include "treet.h"		
11		#include "device.h"		
12		#define MAX_DEV_NAME_LEN 20 /* maximum device name length */		
13		#define MAX_MOD_NAME_LEN 32 /* maximum modeler name length */		
14		extern void		
15		in_host_preproc_info (), /* to set preproc's idea of lines, source name */		
16		in_host_preproc (), /* preprocessor initializer */		
17		extern char		
18		in_host_preproc (), /* preprocessor */		
19		in_host_preproc (), /* called after preproc */		
20		static char buffer [DEV_SIZE] ;		
21		static void		
22		buffer_string (str)		
23		char *str ;		
24		/* fill buf with contents of passed "string" token (strip the quotes) */		
25		{ char quote = '"';		
26		quote = '+' ;		
27		do		
28		{ ++str ;		
29		while (*str != quote)		
30		*str++ = *str++ ;		
31		++str ;		
32		if (*str == quote)		
33		*str++ = *str ;		
34		}		
35		while (*str != '\0') ;		
36		/* need to make sure it fits */		
37		*str = '\0' ;		
38		} /* buffer_string */		
39		char *		
40		in_host_preproc (name , source , dev_name , mod_name , usage)		
41		char		
42		"name",		
43		"source",		
44		"dev_name",		
45		"mod_name",		
46		"usage",		
47		/* make a preliminary lexical pass over the source file text */		
48		/* returns device name, modeler name (if any), and device usage (if private) */		
49		{ char *tok = "" ;		
50		unshort len ;		
51		unshort		
52		dev_name_free = 1 ;		
53		mod_name_free = 1 ;		
54		"dev_name" = NULL ;		
55		"mod_name" = NULL ;		
56		"usage" = 0 ;		
57		in_host_preproc_info (name , source) ;		
58		while ((tok = in_host_preproc ()) != NULL)		
59		{ if (strcmp (tok , "device_name") == 0)		
60		{ if ("dev_name" != NULL)		
61		{ in_host_preproc_info () ;		
62		in_host_preproc		
63		{ 1 ,		
64		/*MSG*/"device_name respecified"		
65		}		
66		tok = in_host_preproc () ;		
67		if (tok != NULL)		
68		{ if (*tok == '+' *tok == '\n')		
69		{ buffer_string (tok) ;		
70		tok = buffer ;		
71		}		
72		len = strlen (tok) ;		
73		if (len == 0)		
74		{ in_host_preproc_info () ;		
75		in_host_preproc		
76		{ 1 ,		
77		/*MSG*/"zero-length device_name illegal"		
78		}		
79		else if (len > MAX_DEV_NAME_LEN)		
80		{ in_host_preproc_info () ;		
81		(void) sprintf (buffer , "%u" , MAX_DEV_NAME_LEN) ;		
82		in_host_preproc		
83		{ 1 ,		
84		/*MSG*/"device_name too long (must be no more than %s characters)"		
85		} ; buffer		
86		}		
87		/* should I allocate the leader of len and MAX_DEV_NAME_LEN and use strcpy? */		
88		"dev_name" = malloc (len + 1) ;		
89		if ("dev_name" == NULL)		
90		{ in_host_preproc		
91		{ 0 ,		
92		/*MSG*/"out of memory on host (device_name)"		
93		}		
94		goto ret_failure ;		
95		(void) strcpy ("dev_name" , tok) ;		
96		dev_name_free = 0 ;		
97		}		
98		}		
99		else if (strcmp (tok , "modeler_name") == 0)		
100		{ if ("mod_name" != NULL)		
101		{ in_host_preproc_info () ;		
102		in_host_preproc		
103		{ 1 ,		
104		/*MSG*/"modeler_name respecified"		
105		}		
106		}		
107		tok = in_host_preproc () ;		

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellsw/Pr_host.c	DATE 5/23/89	PAGE # 2/7
LINE #		SOURCE TEXT		
121		if (tok != NULL)		
122		{ len = strlen (tok) ;		
123		if (len == 0)		
124		{ lm_host_preproc_info () ;		
125		lm_error		
126		{ 1 ,		
127		/*MSG*/"zero-length modular_name illegal"		
128		}		
129		}		
130		else if (len > MAX_MOD_NAME_LEN)		
131		{ lm_host_preproc_info () ;		
132		{ void } sprintf (buffer , "%u" , MAX_MOD_NAME_LEN) ;		
133		lm_error		
134		{ 1 ,		
135		/*MSG*/"modular_name too long (must be no more than %s characters)"		
136		, buffer		
137		}		
138		}		
139		/* should I allocate the lesser of len and MAX_MOD_NAME_LEN and use strcpy? */		
140		*mod_name = malloc ((unsigned) strlen (tok) + 1) ;		
141		if (*mod_name == NULL)		
142		{ lm_error		
143		{ 0 ,		
144		/*MSG*/"out of memory on host (modular_name)"		
145		goto ret_failure ;		
146		}		
147		{ void } strcpy (*mod_name , tok) ;		
148		mod_name_free = 0 ;		
149		}		
150		}		
151		}		
152		else if (strcmp (tok , "device_usage") == 0)		
153		{ if ("usage")		
154		{ lm_host_preproc_info () ;		
155		lm_error		
156		{ 1 ,		
157		/*MSG*/"device_usage respecified"		
158		}		
159		goto ret_failure ;		
160		}		
161		tok = lm_host_preproc () ;		
162		if (tok != NULL)		
163		{ for (t = tok , *t != '\0' ; ++t)		
164		{ if (*t >= 'A' && *t <= 'Z')		
165		*t += 'a' - 'A' ;		
166		*usage = ! strcmp (tok , "private") ;		
167		}		
168		}		
169		}		
170		if (! lm_errors ())		
171		return lm_host_preproc () ;		
172		ret_failure :		
173		if (! dev_name_free)		
174		{ void } free (*dev_name) ;		
175		if (! mod_name_free)		
176		{ void } free (*mod_name) ;		
177		*dev_name = NULL ;		
178		*mod_name = NULL ;		
179		return NULL ;		
180		} /* lm_host_preproc */		
181		/* end of Preprocessor for EML HOST.AIDE */		
182				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89
TIME 4:42:18 pm

PAGE #
1/1

```

1  // SCCS ID: Hconst.c rev 3.1.1, 4/24/89 at 08:00:33
2
3  /*
4   * Constraint Manager for HMM Processor
5   */
6
7  #include "common.h"
8  #include "HML.h"
9  #include "strings.h"
10 #include "strsys.h"
11 #include "spars.h"
12 #include "trees.h"
13 #include "Hconst.h"
14
15 extern void free_dev ( ) /* to keep track of depth of indentation for tracing */
16 static ushort depth; /* the global device structure */
17 struct dev { the; } /* a global character buffer */
18 char buf [ BUF_SIZE ]; /* a global character buffer */
19
20 void
21 init_constraint ( )
22 /* enter all constraint symbols needed by constraint */
23 {
24     (void) strtosym ( "public" );
25     (void) strtosym ( "private" );
26     (void) strtosym ( "hi" );
27     (void) strtosym ( "Hz" );
28     (void) strtosym ( "Hx" );
29     (void) strtosym ( "Cbx" );
30     (void) strtosym ( "as" );
31     (void) strtosym ( "as" );
32     (void) strtosym ( "as" );
33     (void) strtosym ( "infinity" );
34     (void) strtosym ( "static" );
35     (void) strtosym ( "internal" );
36     (void) strtosym ( "external_clocks" );
37     (void) strtosym ( "external_clock1" );
38     (void) strtosym ( "TIL" );
39     (void) strtosym ( "HMS" );
40     (void) strtosym ( "CMS" );
41     (void) strtosym ( "LMS" );
42     (void) strtosym ( "const" );
43     (void) strtosym ( "vac" );
44     (void) strtosym ( "vth" );
45     (void) strtosym ( "vth" );
46     (void) strtosym ( "vth" );
47     (void) strtosym ( "vth" );
48     (void) strtosym ( "vth" );
49     (void) strtosym ( "vth" );
50     (void) strtosym ( "vth" );
51     (void) strtosym ( "vth" );
52     (void) strtosym ( "vth" );
53     (void) strtosym ( "vth" );
54     (void) strtosym ( "eval" );
55     (void) strtosym ( "store" );
56     (void) strtosym ( "edge_rise" );
57     (void) strtosym ( "edge_fall" );
58     (void) strtosym ( "feedback" );
59     (void) strtosym ( "recapitalize" );
60     (void) strtosym ( "in sequence" );
61     (void) strtosym ( "last cycle" );
62     (void) strtosym ( "hard" );
63     (void) strtosym ( "medium" );
64     (void) strtosym ( "hard medium" );
65     (void) strtosym ( "rise" );
66     (void) strtosym ( "fall" );
67     (void) strtosym ( "low" );
68     (void) strtosym ( "high" );
69     (void) strtosym ( "float" );
70     (void) strtosym ( "valid" );
71     (void) strtosym ( "any" );
72     (void) strtosym ( "min" );
73     (void) strtosym ( "typ" );
74     (void) strtosym ( "max" );
75     (void) strtosym ( "as" );
76     (void) strtosym ( "exit" );
77     (void) strtosym ( "missing delays" );
78     (void) strtosym ( "in store changes" );
79     (void) strtosym ( "trace delay" );
80     (void) strtosym ( "pull up" );
81     (void) strtosym ( "pull down" );
82 } /* init_constraint */
83
84 void
85 free_the ( )
86 { free_dev ( the ); }
87 /* free_the */
88
89 /*VARARGS2*/
90 void
91 error ( T, mess, arg, arg2 )
92     tree T;
93     char
94     *mess,
95     *arg,
96     *arg2;
97 /* report a constraint error via lm_Error() */
98 {
99     ASSERT(T!=NULLTREE);
100     lm_Hest_lineno ( line_of ( T ) );
101     lm_Hest_input ( syntostr ( file_of ( T ) ) );
102     lm_Error ( 1, mess, arg, arg2 );
103 } /* error */
104
105 /*VARARGS2*/
106 void
107 warning ( T, mess, arg, arg2 )
108     tree T;
109     char
110     *mess,
111     *arg,
112     *arg2;
113 /* report a constraint warning via lm_Warning() */
114 {
115     ASSERT(T!=NULLTREE);
116     lm_Hest_lineno ( line_of ( T ) );
117     lm_Hest_input ( syntostr ( file_of ( T ) ) );
118     lm_Warning ( mess, arg, arg2 );
119 } /* warning */
120

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89
TIME 4:42:18 pm

PAGE #
2/2

```

LINE #          SOURCE TEXT
121 void
122 buf_instead ( i , j )
123 {
124     /* fill buffer for error reporting */
125     void ) sprintf ( buf , "td instead of td" , i , j ) ;
126     buf_instead "/
127 }
128 void
129 tr_entry ( T )
130 {
131     /* trace entry to a subtree */
132     { ushort i ;
133       if ( lm_st_flag_on ( ) )
134         { for ( i = 0 ; i < depth ; ++i )
135           { void ) lm_print_message ( " " ) ;
136             ( void ) lm_print_message ( "+ " ) ;
137             printnode ( T ) ;
138             ( void ) lm_print_message ( "\n" ) ;
139           }
140         ++depth ;
141       } /* tr_entry */
142     }
143 void
144 tr_exit ( T )
145 {
146     /* trace exit from a subtree */
147     { ushort i ;
148       --depth ;
149       if ( lm_st_flag_on ( ) )
150         { for ( i = 0 ; i < depth ; ++i )
151           { void ) lm_print_message ( " " ) ;
152             ( void ) lm_print_message ( "- " ) ;
153             printnode ( T ) ;
154             ( void ) lm_print_message ( "\n" ) ;
155           }
156       } /* tr_exit */
157     }
158 static char
159 lower ( c )
160 {
161     /* returns the lower case equivalent of the passed character */
162     { return c >= 'A' && c <= 'Z' ? c - 'A' + 'a' : c ;
163     } /* lower */
164 }
165 ushort
166 scompare ( sym1 , sym2 )
167 symbol
168 sym1 ,
169 sym2 ;
170 /* returns 1 iff the two passed symbols' lower case equivalents compare */
171 { char
172   *t1 ,
173   *t2 ;
174   t1 = systostr ( sym1 ) ;
175   t2 = systostr ( sym2 ) ;
176   while ( *t1 != '\0' && *t2 != '\0' && lower ( *t1 ) == lower ( *t2 ) )
177     ++t1 ,
178     ++t2 ;
179   return *t1 == '\0' && *t2 == '\0' ;
180 } /* scompare */
181
182 static struct dev *
183 allo_dev ( )
184 /* returns pointer to newly-allocated device structure */
185 { char *ptr ;
186   ptr = lm_malloc ( sizeof ( struct dev ) ) ;
187   if ( ptr == NULL )
188     { lm_error
189       ( 1 ,
190       /*LMSC*/"out of memory on modeler (allo_dev)"
191       ) ;
192     }
193   lm_fail ( ) ;
194   return ( struct dev * ) ptr ;
195 } /* allo_dev */
196
197 static struct pins *
198 allo_pins ( size )
199 ushort size ;
200 /* returns pointer to newly-allocated array of pin structs of passed size */
201 { char *ptr ;
202   if ( size )
203     ptr = lm_malloc ( sizeof ( struct pins ) * size ) ;
204   else
205     return NULL ;
206   if ( ptr == NULL )
207     { lm_error
208       ( 1 ,
209       /*LMSC*/"out of memory on modeler (allo_pins)"
210       ) ;
211     }
212   lm_fail ( ) ;
213   return ( struct pins * ) ptr ;
214 } /* allo_pins */
215
216 static struct sequence *
217 allo_sequences ( size )
218 ushort size ;
219 /* returns pointer to newly-allocated array of sequences */
220 { char *ptr ;
221   if ( size )
222     ptr = lm_malloc ( sizeof ( struct sequence ) * size ) ;
223   else
224     return NULL ;
225   if ( ptr == NULL )
226     { lm_error
227       ( 1 ,
228       /*LMSC*/"out of memory on modeler (allo_sequences)"
229       ) ;
230     }
231   lm_fail ( ) ;
232   return ( struct sequence * ) ptr ;
233 } /* allo_sequences */
234
235 double
236 check_num ( num )
237 tree num ;
238 /* checks the number whose text is in the passed node */
239 /* returns the floating point value */
240

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hconst.c	DATE, 5/23/89 TIME 4:42:18 pm	PAGE # 3/3
SOURCE TEXT				
LINE #				
241	{ char *str ;			
242	double			
243	val = 0 ;			
244	power = 0 ;			
245	tr_entry (sum) ;			
246	if (nameofnode (sum) == N_NEGATIVE)			
247	{ val = - check_sum (subtree (1 , sum)) ;			
248	tr_exit (sum) ;			
249	return val ;			
250	}			
251	str = systostr (textofnode (sum)) ;			
252	while (*str != '\0')			
253	{ switch (*str)			
254	{ case			
255	{ if (power)			
256	{ error			
257	{ sum			
258	/*LMSC*/"illegal number (multiple decimal points): %s"			
259	, sof (sum)			
260	}			
261	tr_exit (sum) ;			
262	return 0 ;			
263	}			
264	power = 10 ;			
265	break ;			
266	case '0' :			
267	case '1' :			
268	case '2' :			
269	case '3' :			
270	case '4' :			
271	case '5' :			
272	case '6' :			
273	case '7' :			
274	case '8' :			
275	case '9' :			
276	{ if (power)			
277	{ val += (*str - '0') / power ;			
278	power *= 10 ;			
279	}			
280	else			
281	{ val *= 10 ;			
282	val += *str - '0' ;			
283	}			
284	break ;			
285	case '*' :			
286	break ;			
287	default :			
288	error			
289	{ sum			
290	/*LMSC*/"illegal number (non-numeric characters): %s"			
291	, sof (sum)			
292	}			
293	tr_exit (sum) ;			
294	return 0 ;			
295	}			
296	++str ;			
297	}			
298	tr_exit (sum) ;			
299	return val ;			
300	} /* check_sum */			
301				
302	unsigned long			
303	check_int (sum)			
304	tree sum ;			
305	/* checks the number in the passed node via check_sum() */			
306	/* makes sure the value is integral */			
307	/* returns the unsigned long value */			
308	{ double val ;			
309	val = check_sum (sum) ;			
310	if (val - (unsigned long) val != 0)			
311	error			
312	{ sum			
313	/*LMSC*/"integer value required: %s"			
314	, sof (sum)			
315	}			
316	return val ;			
317	} /* check_int */			
318				
319	ushort			
320	do_pin_name (T)			
321	tree T ;			
322	/* processes a pin name, returning the number of pins it indicates */			
323	{ tree kid ;			
324	ushort			
325	sum1 = 0 ;			
326	sum2 = 0 ;			
327	tr_entry (T) ;			
328	switch (nameofnode (T))			
329	{ case P_ID :			
330	case P_NUM :			
331	case P_STR :			
332	/* what about generators in strings? */			
333	break			
334	case M_PIN_NAME :			
335	kid = subtree (2 , T) ;			
336	decorate (kid , (long) (sum1 = sum2 = check_int (kid))) ;			
337	if (kidcount (T) == 3)			
338	{ kid = subtree (3 , T) ;			
339	decorate (kid , (long) (sum1 = check_int (kid))) ;			
340	}			
341	break ;			
342	default :			
343	error			
344	{ T			
345	/*LMSC*/"internal error: do_pin_name: unknown node name: %s"			
346	, systostr (nameofnode (T))			
347	}			
348	break ;			
349	}			
350	tr_exit (T) ;			
351	if (sum1 > sum2)			
352	return sum1 - sum2 + 1 ;			
353	return sum2 - sum1 + 1 ;			
354	} /* do_pin_name */			
355				
356	void			
357	buf_string (str)			
358	char *str ;			
359	/* fill buf with contents of passed "string" token (strip the quotes) */			
360	{ char quote , *b ;			

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM shellswm/Hconst.c	DATE 5/23/89	PAGE #
		TIME 4:42:18 pm	4/4

LINE #	SOURCE TEXT
361	quote = *str ;
362	b = buf ;
363	do
364	{ ++str ;
365	while (*str != quote)
366	*b++ = *str++ ;
367	++str ;
368	if (*str == quote)
369	*b++ = *str ;
370	}
371	while (*str != '\0') ;
372	/* need to make sure it fits */
373	*b = '\0' ;
374	} /* buf_string */
375	
376	symbol
377	prepend_underscore (sym , look_only)
378	symbol sym ;
379	ushort look_only ;
380	/* prepends '_' to the passed symbol, returns the symbol it makes */
381	/* if look_only is nonzero, only returns the symbol it makes if it exists */
382	{ (void) sprintf (buf , "%s", symtostr (sym)) ;
383	/* need to make sure it fits */
384	return look_only ? strlook (buf) : strtosym (buf) ;
385	} /* prepend_underscore */
386	
387	symbol
388	gen_name (pre , num)
389	symbol pre ;
390	ushort num ;
391	/* takes a symbol and a number, returns their concatenation as a symbol */
392	{ char *str ;
393	ushort len ;
394	str = symtostr (pre) ;
395	if (*str == '\0' *str == ...)
396	{ buf_string (str) ;
397	len = strlen (buf) ;
398	(void) sprintf (buf + len , "%u", num) ;
399	/* need to make sure it fits */
400	}
401	else
402	{ (void) sprintf (buf , "%s%u", str , num) ;
403	/* need to make sure it fits */
404	return strtosym (buf) ;
405	} /* gen_name */
406	
407	void
408	add_attr (T , attrs , ndex)
409	tree T ;
410	struct pla_attr *attrs ;
411	ushort ndex ;
412	/* fills in attribute structure at ndex position of attrs */
413	/* with information derived from passed tree T */
414	{ ushort i ;
415	{ tree kid ;
416	symbol
417	text ;
418	text2 ;
419	name ;
420	long ohms ;
421	name = nameofnode (T) ;
422	for (i = 0 ; i < ndex ; ++i)
423	switch (name)
424	{ case P_ID :
425	text = textofnode (T) ;
426	if
427	{ text == C_eval
428	text == C_store
429	text == C_edge_rise
430	text == C_edge_fall
431	}
432	{ if (attrs [i] . use_flag == ID_ATTR)
433	{ text2 = attrs [i] . attr . id ;
434	if
435	{ text2 == C_eval
436	text2 == C_store
437	text2 == C_edge_rise
438	text2 == C_edge_fall
439	}
440	error
441	{ T ,
442	/*MSG*/"%s attribute respecified"
443	, symtostr (text)
444);
445	}
446	}
447	else if (text == C_feedback text == C_keepalive)
448	{ if (attrs [i] . use_flag == ID_ATTR)
449	{ text2 = attrs [i] . attr . id ;
450	if (text2 == text)
451	error
452	{ T ,
453	/*MSG*/"%s attribute respecified"
454	, symtostr (text)
455);
456	}
457	}
458	else if (text == C_pull_up text == C_pull_down)
459	if (attrs [i] . use_flag == ID_ATTR)
460	{ text2 = attrs [i] . attr . id ;
461	if (text2 == C_pull_up text2 == C_pull_down)
462	error
463	{ T ,
464	/*MSG*/"%s attribute respecified"
465	, symtostr (text)
466);
467	}
468	else if (attrs [i] . use_flag == CURRENT_SPEC)
469	{ text2 = attrs [i] . attr . cur_spec . cur ;
470	if (text2 == C_pull_up text2 == C_pull_down)
471	error
472	{ T ,
473	/*MSG*/"%s attribute respecified"
474	, symtostr (text)
475);
476	}
477	break ;
478	case N_ATTR :
479	if (nameofnode (subtree (2 , T)) == P_ID)
480	{ if

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89 PAGE #
TIME 4:42:18 pm 5/5

```

LINE #
481      ( attr { 1 } . use_flag == CYCLE_SPEC &&
482      attr { 1 } . attr . cyc_spec . seq ==
483      ( text = textofnode ( kid = subtree ( 1 , T ) ) )
484      )
485      {
486      ( kid ,
487      /*MSG*/"a attribute respecified"
488      , syntestr ( text )
489      )
490      break ;
491      }
492      text = textofnode ( kid = subtree ( 1 , T ) ) ;
493      if ( text == C_pull_up || text == C_pull_down )
494      {
495      if ( attr { 1 } . use_flag == ID_ATTR )
496      {
497      text2 = attr { 1 } . attr . id ;
498      if ( text2 == C_pull_up || text2 == C_pull_down )
499      {
500      /*MSG*/"a attribute respecified"
501      , syntestr ( text )
502      )
503      break ;
504      }
505      else if ( attr { 1 } . use_flag == CURRENT_SPEC )
506      {
507      text2 = attr { 1 } . attr . cur_spec . cur ;
508      if ( text2 == C_pull_up || text2 == C_pull_down )
509      {
510      /*MSG*/"a attribute respecified"
511      , syntestr ( text )
512      )
513      break ;
514      }
515      }
516      if ( attr { 1 } . use_flag == CURRENT_SPEC &&
517      attr { 1 } . attr . cur_spec . cur == text
518      )
519      {
520      /*MSG*/"a attribute respecified"
521      , syntestr ( text )
522      )
523      break ;
524      }
525      default :
526      {
527      /*MSG*/"internal error: add_attr: bad attribute"
528      , syntestr ( text )
529      )
530      break ;
531      }
532      switch ( name )
533      {
534      case P_ID :
535      {
536      attr { ndex } . use_flag = ID_ATTR ;
537      attr { ndex } . attr . id = textofnode ( T ) ;
538      break ;
539      }
540      case H_ATTR :
541      {
542      if ( nameofnode ( subtree ( 2 , T ) ) == P_ID )
543      {
544      attr { ndex } . use_flag = CYCLE_SPEC ;
545      attr { ndex } . attr . cyc_spec . seq =
546      textofnode ( subtree ( 1 , T ) ) ;
547      attr { ndex } . attr . cyc_spec . drv =
548      textofnode ( subtree ( 2 , T ) ) ;
549      break ;
550      }
551      attr { ndex } . use_flag = CURRENT_SPEC ;
552      text2 =
553      attr { ndex } . attr . cur_spec . cur =
554      textofnode ( subtree ( 1 , T ) ) ;
555      if ( text2 == C_pull_up || text2 == C_pull_down )
556      {
557      attr { ndex } . attr . cur_spec . val =
558      ohms = dec_ratio ( subtree ( 2 , T ) ) ;
559      if ( text2 == C_pull_up )
560      {
561      if ( ( the -> Vcc / ohms * 2000 > 4 )
562      )
563      {
564      /*MSG*/"pull_up resistance too low: %s"
565      , syntestr ( textofnode ( subtree ( 2 , T ) ) )
566      )
567      break ;
568      }
569      }
570      else if ( ( the -> Vcc / ohms * 2000 > 4 )
571      )
572      {
573      /*MSG*/"pull_down resistance too low: %s"
574      , syntestr ( textofnode ( subtree ( 2 , T ) ) )
575      )
576      break ;
577      }
578      attr { ndex } . attr . cur_spec . val =
579      textofnode ( subtree ( 1 , T ) ) == C_Yil ||
580      textofnode ( subtree ( 1 , T ) ) == C_Ioh ?
581      -check_sum ( subtree ( 2 , T ) ) :
582      check_sum ( subtree ( 2 , T ) ) ;
583      break ;
584      }
585      default :
586      {
587      /*MSG*/"internal error: add_attr: unknown node name: %s"
588      , syntestr ( text )
589      )
590      break ;
591      }
592      }
593      }
594      /* add_attr */
595      static void
596      init_device ( t )
597      struct dev *t ;
598      /* initializes all fields of the passed device structure */
599      {
600      t -> edges =
601      t -> num_in =
602      t -> num_out =
603      t -> num_io =
604      t -> num_power =
605      t -> num_grounded =
606      t -> num_ac =
607      t -> pin_cat =
608      t -> apin_cat =
609      t -> reset_cat =
610      t -> num_seq =
611      t -> rise_fb =
612      t -> pre_seq_length =

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89
TIME 4:42:18 pm

PAGE #
6/6

```

LINE # SOURCE TEXT
601 t->fb_seq_length =
602 t->post_seq_length =
603 t->ultra_fast =
604 t->dis_tlm_check =
605 t->inh_tlm_measure =
606 t->delays_name =
607 t->use_default =
608 t->has_store_pins =
609 t->has_adapter_map =
610 t->has_package_map =
611 t->has_feedback_pins =
612 0;
613 t->clk_period1 =
614 t->clk_period2 =
615 t->min_default =
616 t->typ_default =
617 t->max_default =
618 -1;
619 t->dev_name =
620 t->dev_type =
621 t->mod_name =
622 t->clk_type =
623 t->technology =
624 t->missing_delays =
625 t->io_store_changes =
626 NULLSYM;
627 t->tech_line = NULLTREE;
628 t->def_setup =
629 t->def_hold =
630 t->def_sample =
631 t->vcc =
632 t->vth =
633 t->vth =
634 t->vih =
635 t->vil =
636 t->vah =
637 t->val =
638 t->ioh =
639 t->iol =
640 t->iih =
641 t->iil =
642 t->ish =
643 t->ial =
644 UNDEFINED;
645 t->pins = NULL;
646 t->sequences = NULL;
647 /* init_device */
648
649 static double
650 round_up ( val , w1 , w2 , w3 , w4 )
651 double
652 val ,
653 w1 ,
654 w2 ,
655 w3 ,
656 w4 ;
657 /* returns val rounded up to something representable with passed weights */
658 {
659 double
660 val = 0.0005 ;
661 ret = w1 + w2 + w3 + w4 ;
662 if ( ret - w4 >= val )
663     ret -= w4 ;
664 if ( ret - w3 >= val )
665     ret -= w3 ;
666 if ( ret - w2 >= val )
667     ret -= w2 ;
668 if ( ret - w1 >= val )
669     ret -= w1 ;
670 return ret ;
671 } /* round_up */
672
673 void init_the ( )
674 { the = NULL ;
675   /* init_the */
676 }
677
678 struct dev *
679 constrain ( T )
680 tree T ;
681 /* perform static semantic checking on the HNL specification */
682 { ushort
683   kids ,
684   pin_of ,
685   pin_seen ,
686   symbol ,
687   sum ,
688   pin ;
689 #ifdef ENFORCE_MARGINS
690 double noise_margin ;
691 char sum_buf [ 12 ] ;
692 #endif ENFORCE_MARGINS
693 depth = 0 ;
694 if
695   ( lm_Rt_flag_on ( ) &&
696     ( lm_Rl_flag_on ( ) || lm_Sp_flag_on ( ) || lm_Ss_flag_on ( ) )
697   )
698   ( void ) lm_print_message ( "\n" ) ;
699
700 t: allo_dev ( ) ;
701 init_device ( the ) ;
702 kids = kidcount ( T ) ;
703 tr_entry ( T ) ;
704
705 /* in walk_1 we:
706 process:
707 device_name
708 ultra_fast
709 device_usage
710 report
711 modeler_name
712 clock_type
713 disable_timing_checking
714 device_hold_time
715 device_setup_time
716 device_sample_time
717 device_speed
718 technology
719 Vcc/Vth/Vih/Vil/Vah/Vsl specification
720 Ioh/Iol/Iih/Iil specification
721 default_delay

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89
TIME 4:42:18 pm

PAGE #
7/7

```

LINE #          SOURCE TEXT
721      count in/out/in/power/ground/ac pin names
722      check against number of pin numbers
723      count pre/fin/post reset sequences
724      process device adapter mapping
725      declare device adapter pin names
726
727      for ( k = 1 ; k <= kids ; ++k )
728          walk_1 ( subtree ( k , T ) ) ;
729      if ( the -> has_adapter_map == 0 )
730          error
731              ( T ,
732                /*MSG*/"no adapter mapping specified"
733              ) ;
734
735      ( /* begin device-level default processing */
736        if ( the -> dev_name == NULLSYM )
737            error
738                ( T ,
739                  /*MSG*/"internal error: no device_name found"
740                ) ;
741        if ( the -> dev_type == NULLSYM )
742            the -> dev_type = C_public ;
743        if ( the -> clk_period == -1 )
744            error
745                ( T ,
746                  /*MSG*/"no device speed specified"
747                ) ;
748        if ( the -> clk_type == NULLSYM )
749            the -> clk_type = C_internal ;
750        switch ( the -> technology )
751        { case C_custom :
752            if ( the -> Vcc == UNDEFINED )
753                error
754                    ( the -> tech_tree ,
755                      /*MSG*/"Vcc undefined"
756                    ) ;
757            if ( the -> Vth == UNDEFINED )
758                error
759                    ( the -> tech_tree ,
760                      /*MSG*/"Vth undefined"
761                    ) ;
762            if ( the -> Vih == UNDEFINED )
763                error
764                    ( the -> tech_tree ,
765                      /*MSG*/"Vih undefined"
766                    ) ;
767            if ( the -> Vil == UNDEFINED )
768                error
769                    ( the -> tech_tree ,
770                      /*MSG*/"Vil undefined"
771                    ) ;
772            if ( the -> Vih == UNDEFINED )
773                error
774                    ( the -> tech_tree ,
775                      /*MSG*/"Vih undefined"
776                    ) ;
777            if ( the -> Vil == UNDEFINED )
778                error
779                    ( the -> tech_tree ,
780                      /*MSG*/"Vil undefined"
781                    ) ;
782            if ( the -> Val == UNDEFINED )
783                error
784                    ( the -> tech_tree ,
785                      /*MSG*/"Val undefined"
786                    ) ;
787            if ( the -> Ioh == UNDEFINED )
788                error
789                    ( the -> tech_tree ,
790                      /*MSG*/"Ioh undefined"
791                    ) ;
792            if ( the -> Iol == UNDEFINED )
793                error
794                    ( the -> tech_tree ,
795                      /*MSG*/"Iol undefined"
796                    ) ;
797            if ( the -> Iih == UNDEFINED )
798                error
799                    ( the -> tech_tree ,
800                      /*MSG*/"Iih undefined"
801                    ) ;
802            if ( the -> Iil == UNDEFINED )
803                error
804                    ( the -> tech_tree ,
805                      /*MSG*/"Iil undefined"
806                    ) ;
807            /* fall through */
808            case NULLSYM :
809                the -> technology = C_TTL ;
810                /* fall through */
811            default :
812                /* fall through */
813            case C_TTL :
814                if ( the -> Vcc == UNDEFINED )
815                    the -> Vcc = D_TTL_Vcc ;
816                if ( the -> Vth == UNDEFINED )
817                    the -> Vth = D_TTL_Vth ;
818                if ( the -> Vih == UNDEFINED )
819                    the -> Vih = D_TTL_Vih ;
820                if ( the -> Vil == UNDEFINED )
821                    the -> Vil = D_TTL_Vil ;
822                if ( the -> Vih == UNDEFINED )
823                    the -> Vih = D_TTL_Vih ;
824                if ( the -> Vil == UNDEFINED )
825                    the -> Vil = D_TTL_Vil ;
826                if ( the -> Val == UNDEFINED )
827                    the -> Val = D_TTL_Val ;
828                if
829                (
830                    the -> Iih == UNDEFINED &&
831                    the -> Iib == UNDEFINED &&
832                    the -> Iol == UNDEFINED &&
833                    the -> Iob == UNDEFINED
834                )
835                {
836                    the -> Iih = ( the -> Iob = the -> Isl = D_TTL_Isl ) / 2 ;
837                    the -> Iib = ( the -> Iol = the -> Ish = D_TTL_Ish ) / 2 ;
838                }
839            else
840            {
841                if ( the -> Iob == UNDEFINED )
842                    the -> Iob = D_TTL_Isl ;
843                if ( the -> Iol == UNDEFINED )
844                    the -> Iol = D_TTL_Ish ;
845            }
846        }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hconst.c

DATE 5/23/89
TIME 4:42:18 pm

PAGE #
8/8

```

LINE #
841     the -> Iol = D_TTL_Ish ;
842     if ( the -> Iih == UNDEFINED )
843     the -> Iih = D_TTL_Ish / 2 ;
844     if ( the -> Iil == UNDEFINED )
845     the -> Iil = D_TTL_Ial / 2 ;
846
847     break ;
848 case C_NMOS :
849     if ( the -> Vcc == UNDEFINED )
850     the -> Vcc = D_NMOS_Vcc ;
851     if ( the -> Vth == UNDEFINED )
852     the -> Vth = D_NMOS_Vth ;
853     if ( the -> Vih == UNDEFINED )
854     the -> Vih = D_NMOS_Vih ;
855     if ( the -> Vil == UNDEFINED )
856     the -> Vil = D_NMOS_Vil ;
857     if ( the -> Voh == UNDEFINED )
858     the -> Voh = D_NMOS_Voh ;
859     if ( the -> Vol == UNDEFINED )
860     the -> Vol = D_NMOS_Val ;
861
862     if
863     (
864         the -> Iil == UNDEFINED &&
865         the -> Iih == UNDEFINED &&
866         the -> Iol == UNDEFINED &&
867         the -> Ioh == UNDEFINED
868     )
869     {
870         the -> Iil = ( the -> Ioh = the -> Iol = D_NMOS_Ial ) / 2 ;
871         the -> Iih = ( the -> Iol = the -> Ish = D_NMOS_Ish ) / 2 ;
872     }
873     else
874     {
875         if ( the -> Ioh == UNDEFINED )
876         the -> Ioh = D_NMOS_Ial ;
877         if ( the -> Iol == UNDEFINED )
878         the -> Iol = D_NMOS_Ish ;
879         if ( the -> Iih == UNDEFINED )
880         the -> Iih = D_NMOS_Ish / 2 ;
881         if ( the -> Iil == UNDEFINED )
882         the -> Iil = D_NMOS_Ial / 2 ;
883     }
884     break ;
885 case C_CMOS :
886     if ( the -> Vcc == UNDEFINED )
887     the -> Vcc = D_CMOS_Vcc ;
888     if ( the -> Vth == UNDEFINED )
889     the -> Vth = the -> Vcc - 0.3 ;
890     if ( the -> Vih == UNDEFINED )
891     the -> Vih = the -> Vcc + 0.70 ;
892     if ( the -> Vil == UNDEFINED )
893     the -> Vil = the -> Vcc + 0.20 ;
894     if ( the -> Voh == UNDEFINED )
895     the -> Voh = the -> Vcc - 0.5 ;
896     if ( the -> Vol == UNDEFINED )
897     the -> Vol = D_CMOS_Val ;
898
899     if
900     (
901         the -> Iil == UNDEFINED &&
902         the -> Iih == UNDEFINED &&
903         the -> Iol == UNDEFINED &&
904         the -> Ioh == UNDEFINED
905     )
906     {
907         the -> Iil = ( the -> Ioh = the -> Ial = D_CMOS_Ial ) / 2 ;
908         the -> Iih = ( the -> Iol = the -> Ish = D_CMOS_Ish ) / 2 ;
909     }
910     else
911     {
912         if ( the -> Ioh == UNDEFINED )
913         the -> Ioh = D_CMOS_Ial ;
914         if ( the -> Iol == UNDEFINED )
915         the -> Iol = D_CMOS_Ish ;
916         if ( the -> Iih == UNDEFINED )
917         the -> Iih = D_CMOS_Ish / 2 ;
918         if ( the -> Iil == UNDEFINED )
919         the -> Iil = D_CMOS_Ial / 2 ;
920     }
921     break ;
922 case C_LCMOS :
923     if ( the -> Vcc == UNDEFINED )
924     the -> Vcc = D_LCMOS_Vcc ;
925     if ( the -> Vth == UNDEFINED )
926     the -> Vth = the -> Vcc - 0.3 ;
927     if ( the -> Vih == UNDEFINED )
928     the -> Vih = D_LCMOS_Vih ;
929     if ( the -> Vil == UNDEFINED )
930     the -> Vil = the -> Vcc + 0.75 ;
931     if ( the -> Voh == UNDEFINED )
932     the -> Voh = the -> Vcc + 0.18 ;
933     if ( the -> Vol == UNDEFINED )
934     the -> Vol = D_LCMOS_Val ;
935
936     if
937     (
938         the -> Iil == UNDEFINED &&
939         the -> Iih == UNDEFINED &&
940         the -> Iol == UNDEFINED &&
941         the -> Ioh == UNDEFINED
942     )
943     {
944         the -> Iil = ( the -> Ioh = the -> Ial = D_LCMOS_Ial ) / 2 ;
945         the -> Iih = ( the -> Iol = the -> Ish = D_LCMOS_Ish ) / 2 ;
946     }
947     else
948     {
949         if ( the -> Ioh == UNDEFINED )
950         the -> Ioh = D_LCMOS_Ial ;
951         if ( the -> Iol == UNDEFINED )
952         the -> Iol = D_LCMOS_Ish ;
953         if ( the -> Iih == UNDEFINED )
954         the -> Iih = D_LCMOS_Ish / 2 ;
955         if ( the -> Iil == UNDEFINED )
956         the -> Iil = D_LCMOS_Ial / 2 ;
957     }
958     break ;
959
960 #ifdef ENFORCE_MARGINS
961     if ( the -> Vth > the -> Vcc - 0.05 )
962     {
963         /* MSG: "illegal Vth specification (must be 50 mV or more below Vcc)"
964         */
965     }
966 #endif ENFORCE_MARGINS

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hconst.c	DATE 5/23/89	PAGE # 9/9
LINE #		SOURCE TEXT		
961		if (the -> Vth > the -> Vcc)		
962		error		
963		{ subtree (1, the -> tech_tree) ,		
964		/*IMSG*/"illegal Vth specification (must be no more than Vcc)"		
965		{		
966		if (the -> Vth < the -> Vcc / 2)		
967		error		
968		{ subtree (1, the -> tech_tree) ,		
969		/*IMSG*/"illegal Vth specification (must be no less than 50% of Vcc)"		
970		{		
971		# ifdef ENFORCE_MARGINS		
972		if (the -> Vth > the -> Vth - 0.05)		
973		error		
974		{ subtree (1, the -> tech_tree) ,		
975		/*IMSG*/"illegal Vth specification (must be 50 mV or more below Vth)"		
976		{		
977		# endif ENFORCE_MARGINS		
978		if (the -> Vth > the -> Vcc)		
979		error		
980		{ subtree (1, the -> tech_tree) ,		
981		/*IMSG*/"illegal Vth specification (must be no more than Vcc)"		
982		{		
983		if (the -> Vth < the -> Vcc / 2)		
984		error		
985		{ subtree (1, the -> tech_tree) ,		
986		/*IMSG*/"illegal Vth specification (must be no less than 50% of Vcc)"		
987		{		
988		if (the -> Vth > the -> Vcc)		
989		error		
990		{ subtree (1, the -> tech_tree) ,		
991		/*IMSG*/"illegal Vth specification (must be no more than Vcc)"		
992		{		
993		# ifdef ENFORCE_MARGINS		
994		noise_margin =		
995		the -> technology == C_LCMS the -> technology == C_custom ?		
996		LCMS_NOISE_MARGIN : NOISE_MARGIN ,		
997		if (the -> Vth > the -> Vth - noise_margin + 0.000001)		
998		{ void sprintf (num_buf , "%g" , the -> Vth - noise_margin) ,		
999		error		
1000		{ subtree (1, the -> tech_tree) ,		
1001		/*IMSG*/"illegal Vth specification (must be no more than 1s Volts)"		
1002		{ num_buf		
1003		{		
1004		}		
1005		if (the -> Vth > the -> Vth - 0.05)		
1006		error		
1007		{ subtree (1, the -> tech_tree) ,		
1008		/*IMSG*/"illegal Vth specification (must be 50 mV or more below Vth)"		
1009		{		
1010		if (the -> Vth > the -> Vth - noise_margin + 0.000001)		
1011		{ void sprintf (num_buf , "%g" , the -> Vth - noise_margin) ,		
1012		error		
1013		{ subtree (1, the -> tech_tree) ,		
1014		/*IMSG*/"illegal Vth specification (must be no less than 1s Volts)"		
1015		{ num_buf		
1016		{		
1017		}		
1018		if (the -> Vth > the -> Vth - 0.05)		
1019		error		
1020		{ subtree (1, the -> tech_tree) ,		
1021		/*IMSG*/"illegal Vth specification (must be 50 mV or more above Vth)"		
1022		{		
1023		# endif ENFORCE_MARGINS		
1024		if (the -> Vth > the -> Vcc / 5)		
1025		error		
1026		{ subtree (1, the -> tech_tree) ,		
1027		/*IMSG*/"illegal Vth specification (must be no more than 20% of Vcc)"		
1028		{		
1029		# ifdef ENFORCE_MARGINS		
1030		if (the -> Vth < 0.05)		
1031		error		
1032		{ subtree (1, the -> tech_tree) ,		
1033		/*IMSG*/"illegal Vth specification (must be no less than 50 mV)"		
1034		{		
1035		# endif ENFORCE_MARGINS		
1036		if (the -> Vth > the -> Vcc / 5)		
1037		error		
1038		{ subtree (1, the -> tech_tree) ,		
1039		/*IMSG*/"illegal Vth specification (must be no more than 20% of Vcc)"		
1040		{		
1041		if (the -> Ial == UNDEFINED)		
1042		{ the -> Ial = 1.9 * the -> Iil ;		
1043		the -> Ial = round_up (the -> Ial , 0.2 , 0.5 , 1.2 , 3.0) ;		
1044		the -> Iah = 1.9 * the -> Iih ;		
1045		the -> Iah = round_up (the -> Iah , 0.1 , 0.5 , 1.2 , 2.0) ;		
1046		{		
1047		# ifdef ENFORCE_CURRENTS		
1048		if (the -> Iah + 0.000001 < 2 * the -> Iil)		
1049		error		
1050		{ subtree (1, the -> tech_tree) ,		
1051		/*IMSG*/"Iah must be at least twice Iil"		
1052		{		
1053		if (the -> Ial + 0.000001 < 2 * the -> Iih)		
1054		error		
1055		{ subtree (1, the -> tech_tree) ,		
1056		/*IMSG*/"Ial must be at least twice Iih"		
1057		{		
1058		# endif ENFORCE_CURRENTS		
1059		if (the -> sum_in == 0 && the -> sum_io == 0)		
1060		error		
1061		{ T ,		
1062		/*IMSG*/"no in_pin or io_pin specified"		
1063		{		
1064		if (the -> sum_out == 0 && the -> sum_io == 0)		
1065		error		
1066		{ T ,		
1067		/*IMSG*/"no out_pin or io_pin specified"		
1068		{		
1069		if (! the -> has_store_pins)		
1070		error		
1071		{ T ,		
1072		/*IMSG*/"no eval pins or store pins specified"		
1073		{		
1074		if (the -> sum_power == 0)		
1075		cwarning		
1076		{ T ,		
1077		/*IMSG*/"no power_pin specified"		
1078		{		
1079		if (the -> sum_ground == 0)		
1080		cwarning		

Copyright 1989 Logic Modeling Systems	SOURCE PROGRAM shellswm/Hconst.c	DATE 5/23/89 TIME 4:42:18 pm	PAGE # 10/10
LINE #	SOURCE TEXT		
1081	/* T, */		
1082	/*IMSG*/"no ground pin specified"		
1083	/* end device-level defaults processing */		
1084	/* walk 2 we:		
1085	process package mapping		
1086	declare pin numbers		
1087	/*		
1088	for (k = 1 ; k <= kids ; ++k)		
1089	walk_2 (subtree (k , T)) ;		
1090	if (the -> has_package_map == 0)		
1091	error		
1092	/* T, */		
1093	/*IMSG*/"no package mapping specified"		
1094	/*		
1095	/* in walk 3 we:		
1096	fill pin array and attribute arrays		
1097	declare pin names		
1098	/*		
1099	pin_seen =		
1100	the -> num_in + the -> num_out + the -> num_io +		
1101	the -> num_reset + the -> num_ground + the -> num_sc ;		
1102	the -> pins = allo_pins (pin_seen) ;		
1103	for (k = 0 ; k < pin_seen ; ++k)		
1104	{ the -> pins [k] . attr = NULL ;		
1105	the -> pins [k] . timing_list = NULL ;		
1106	}		
1107	for (k = 1 ; k <= kids ; ++k)		
1108	walk_3 (subtree (k , T)) ;		
1109	ASSERT(the->pin_cnt==pin_seen);		
1110	/* in walk 4 we:		
1111	process initialization sequences		
1112	process delays		
1113	process pin name mappings		
1114	/*		
1115	if (the -> has_feedback_pins && ! the -> num_seqs)		
1116	error		
1117	/* T, */		
1118	/*IMSG*/"must use as initialization sequence to have a feedback signal"		
1119	/*		
1120	the -> sequences = allo_sequences (the -> num_seqs) ;		
1121	for (k = 1 ; k <= kids ; ++k)		
1122	walk_4 (subtree (k , T)) ;		
1123	ASSERT(the->reset_cnt==the->num_seqs);		
1124	/* assign pin numbers */		
1125	if (! is_errors ())		
1126	for (k = 0 ; k < the -> pin_cnt ; ++k)		
1127	{ (void) sprintf (buf , "%d" , k) ;		
1128	num = strtok (buf , " ") ;		
1129	pin_seen = 0 ;		
1130	while		
1131	{ num != NULLSYM && (pin = pin_number_of (num)) != NULLSYM		
1132	{ if		
1133	{ (pin = pin_name_of (pin)) != NULLSYM &&		
1134	{ (pin = pin_number_of (pin)) != NULLSYM		
1135	}		
1136	if (pin_seen)		
1137	{ (void) sprintf (buf , "%d" , k) ;		
1138	error		
1139	/* T, */		
1140	/*IMSG*/"device adapter pin number is re-used by device signal name is"		
1141	buf ,		
1142	systrcat (pin)		
1143	/*		
1144	else		
1145	{ pin_seen = 1 ;		
1146	pin_of = pin_name_of (pin) ;		
1147	while (the -> pins [pin_of] . number < MAX_PINS)		
1148	++pin_of ;		
1149	the -> pins [pin_of] . number = k ;		
1150	}		
1151	num = prepend_underscore (num , 1) ;		
1152	}		
1153	if (is_errors ())		
1154	{ strfree () ;		
1155	treefree () ;		
1156	free_the () ;		
1157	return NULL ;		
1158	}		
1159	return the ;		
1160	/* constrain */		
1161	/* end of Constrainer for WML Processor */		
1162			
1163			
1164			
1165			
1166			
1167			
1168			
1169			
1170			
1171			
1172			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89
TIME 4:42:19 pm

PAGE #
1/11

SOURCE TEXT

```

1  /* SCCS_ID: Hgen.c rev 3.1.1, 4/24/89 at 08:00:37 */
2
3  /*
4   * Generator for HDGL Processor
5   */
6
7  #include "common.h"
8  #include "HDGL.h"
9  #include "strings.h"
10 #include "string.h"
11 #include "space.h"
12 #include "trees.h"
13 #include "hconst.h"
14 #include "device.h"
15
16 #ifdef WRITE_DEVICE
17 #include <stdio.h>
18 #endif WRITE_DEVICE
19
20 #define EDGE_SETUP_TIME 5000 /* picoseconds */
21 #define MD_SETTLING_TIME 15000 /* picoseconds */
22 #define ED_SETTLING_TIME 0 /* picoseconds */
23
24 extern void free_device ( ) ;
25
26 static ushort zero = 0 ; /* used in ASSERTions */
27 static unsigned long
28 #ifdef WRITE_DEVICE
29 offset , /* used in writing device structure */
30 #endif WRITE_DEVICE
31 text_size , /* size of text for writing device structure */
32 struct device *device ;
33
34 static DEVICE_SPEC *
35 allo_device ( )
36 /* returns pointer to newly-allocated device structure */
37 {
38     DEVICE_SPEC *ptr ;
39     ptr = ( DEVICE_SPEC * ) lm_malloc ( sizeof ( DEVICE_SPEC ) ) ;
40     if ( ptr == NULL )
41         { lm_Error
42           ( 1 ,
43             /*MSG*/"out of memory on modeler (allo_device)"
44           ) ;
45           lm_Fail ( ) ;
46         }
47     ptr->device_name = NULL ;
48     ptr->modeler_name = NULL ;
49     ptr->pin_table = NULL ;
50     ptr->net_table = NULL ;
51     ptr->timing_table = NULL ;
52     ptr->ntyp_table = NULL ;
53     return ptr ;
54 } /* allo_device */
55
56 static PIN_SPEC *
57 allo_pins ( count )
58 /* returns pointer to newly-allocated pin structure array of passed size */
59 {
60     PIN_SPEC *ptr , *p ;
61     if ( count )
62         ptr = ( PIN_SPEC * ) lm_malloc ( sizeof ( PIN_SPEC ) * count ) ;
63     else
64         return NULL ;
65     if ( ptr == NULL )
66         { lm_Error
67           ( 1 ,
68             /*MSG*/"out of memory on modeler (allo_pins)"
69           ) ;
70           lm_Fail ( ) ;
71         }
72     p = ptr ;
73     while ( count-- )
74     { p->pin_name = NULL ;
75       p->pin_number = NULL ;
76       p->pin_alias = NULL ;
77       p->direction = NONE ;
78       ++p ;
79     }
80     return ptr ;
81 } /* allo_pins */
82
83 static TIMING_SPEC *
84 allo_timings ( count )
85 /* returns pointer to newly-allocated timing structure array */
86 {
87     TIMING_SPEC *ptr ;
88     if ( count )
89         ptr = ( TIMING_SPEC * ) lm_malloc ( sizeof ( TIMING_SPEC ) * count ) ;
90     else
91         return NULL ;
92     if ( ptr == NULL )
93         { lm_Error
94           ( 1 ,
95             /*MSG*/"out of memory on modeler (allo_timings)"
96           ) ;
97           lm_Fail ( ) ;
98         }
99     return ptr ;
100 } /* allo_timings */
101
102 static MIN_TYP_MAX *
103 allo_ntyps ( count )
104 /* returns pointer to newly-allocated min-tyt-max structure array */
105 {
106     MIN_TYP_MAX *ptr ;
107     if ( count )
108         ptr = ( MIN_TYP_MAX * ) lm_malloc ( sizeof ( MIN_TYP_MAX ) * count ) ;
109     else
110         return NULL ;
111     if ( ptr == NULL )
112         { lm_Error
113           ( 1 ,
114             /*MSG*/"out of memory on modeler (allo_ntyps)"
115           ) ;
116           lm_Fail ( ) ;
117         }
118     return ptr ;
119 } /* allo_ntyps */
120 static SEO_SPEC *

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hgen.c	DATE 5/23/89	PAGE # 2/12
LINE #		SOURCE TEXT		
121		/* alloc_seqs (count)		
122		/* returns pointer to newly-allocated sequence structure array */		
123		{ SEQ_SPEC *ptr , *p ;		
124		if (count)		
125		ptr = (SEQ_SPEC *) malloc (sizeof (SEQ_SPEC) * count) ;		
126		else		
127		return NULL ;		
128		if (ptr == NULL)		
129		{		
130		if (!error)		
131		{		
132		/*LMSG*/out of memory on modeler (alloc_seqs)		
133		{		
134		if (!error)		
135		{		
136		p = ptr ;		
137		while (count--)		
138		{ p->pre_bits = NULL ;		
139		p->fb_bits = NULL ;		
140		p->post_bits = NULL ;		
141		p->seq_str = NULL ;		
142		}		
143		return ptr ;		
144		} /* alloc_seqs */		
145				
146		static char *		
147		alloc_bits (length)		
148		/* returns pointer to newly-allocated bit array of rounded-up length */		
149		{ char *ptr ;		
150		length =		
151		length >= 1 ?		
152		length : 1 ;		
153		ptr = malloc (length) ;		
154		if (ptr == NULL)		
155		{		
156		if (!error)		
157		{		
158		/*LMSG*/out of memory on modeler (alloc_bits)		
159		{		
160		if (!error)		
161		{		
162		text_size += length ;		
163		return ptr ;		
164		} /* alloc_bits */		
165				
166		static char *		
167		alloc_string (sym)		
168		symbol sym ;		
169		{ char *str , *ret ;		
170		ushort len ;		
171		str = symstr (sym) ;		
172		ret = malloc (len = strlen (str) + 1) ;		
173		if (ret == NULL)		
174		{		
175		if (!error)		
176		/*LMSG*/out of memory on modeler (alloc_string)		
177		{		
178		if (!error)		
179		{		
180		text_size += len ;		
181		(void) strcpy (ret , str) ;		
182		return ret ;		
183		} /* alloc_string */		
184				
185		static ushort		
186		is_dup_stym (dev , tim)		
187		struct dev *dev ;		
188		struct timing *tim ;		
189		/* searches timing specs on all pins in constraint device structure */		
190		/* returns 1 iff the passed timing has the same min-typ-max values */		
191		{ ushort i ;		
192		struct timing *t ;		
193		for (i = 0 ; i < dev->pin_cnt ; ++i)		
194		if (dev->pins [i] . number < MAX_PINS)		
195		{ t = dev->pins [i] . timing_list ;		
196		while (t != NULL)		
197		{ if (t == tim)		
198		{		
199		return 0 ;		
200		if (t->minimum == tim->minimum &&		
201		t->typical == tim->typical &&		
202		t->maximum == tim->maximum		
203		{		
204		return 1 ;		
205		t = t->next ;		
206		}		
207		}		
208		ASSERT(zero);		
209		return 0 ;		
210		} /* is_dup_stym */		
211				
212		static ushort		
213		state_of (state)		
214		symbol state ;		
215		/* maps constraint states to device structure states */		
216		{ switch (state)		
217		{ case NULLSYM :		
218		return NONE ;		
219		case C_low :		
220		return LOW ;		
221		case C_high :		
222		return HIGH ;		
223		case C_float :		
224		return FLOAT ;		
225		case C_valid :		
226		return VALID ;		
227		case C_any :		
228		return ANY ;		
229		default :		
230		ASSERT(-zero);		
231		return 0 ;		
232		}		
233		} /* state_of */		
234				
235		static ushort		
236		add_timing (dev , d , tim , timings , stym)		
237		DEVICE_SPEC *dev ;		
238		struct dev *d ;		
239		struct timing *tim ;		
240		ushort		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89 PAGE #
TIME 4:42:19 pm 3/13

```

LINE # SOURCE TEXT
241 timings ,
242 mtyms ,
243 /* add the passed timing structure to the timings in the device structure */
244 { ushort i
245 TIMING_SPEC *t ,
246 MIN_TTP_MAX *mtyms ,
247 t = dev->timings_table + timings ,
248 t->minor_pin =
249 tim->minor_pin_name == NULLSYM ?
250 MAX_PINS :
251 d->pins [ pin_name_of ( tim->minor_pin_name ) ] . number ,
252 t->aspec = 0 ,
253 t->major_state = state_of ( tim->major_state ) ,
254 t->minor_state = state_of ( tim->minor_state ) ,
255 for ( i = 0 ; i < mtyms ; ++i )
256 { mtyms = dev->mtyms_table + i ,
257 if
258 ( mtyms->minimum == tim->minimum &&
259 mtyms->typical == tim->typical &&
260 mtyms->maximum == tim->maximum
261 )
262 { t->mtyms_index = i ,
263 return 0 ,
264 }
265 }
266 t->mtyms_index = mtyms ,
267 mtyms = dev->mtyms_table + mtyms ,
268 mtyms->minimum = tim->minimum ,
269 mtyms->typical = tim->typical ,
270 mtyms->maximum = tim->maximum ,
271 return 1 ,
272 } /* add_timing */
273
274 static ushort
275 valof ( sym )
276 symbol sym ,
277 /* returns the integer value of the passed symbol */
278 { ushort val = 0 ,
279 char *str = symtostr ( sym ) ,
280 do
281 { val *= 10 ,
282 val += *str - '0' ,
283 }
284 while ( ++str != '\0' ) ,
285 return val ,
286 } /* valof */
287
288 static void
289 bitgen ( bits , index_addr , T )
290 char *bits ,
291 ushort *index_addr ,
292 tree T ,
293 /* generates the bit sequence for T into the passed character string */
294 { ushort
295 k ,
296 kida ,
297 count ,
298 switch ( nameofnode ( T ) )
299 { case P_PWM :
300 count = valof ( textofnode ( T ) ) != 0 ,
301 kida = *index_addr + 7 ,
302 if ( kida == 0 )
303 bits [ *index_addr >> 3 ] = count << 7 ,
304 else
305 bits [ *index_addr >> 3 ] |= count << ( 7 - kida ) ,
306 *index_addr ++ ,
307 break ,
308 case M_LIST :
309 kida = kidoount ( T ) ,
310 for ( k = 1 ; k <= kida ; ++k )
311 bitgen ( bits , index_addr , subtree ( k , T ) ) ,
312 break ,
313 case M_REPEAT :
314 count = valof ( textofnode ( subtree ( 1 , T ) ) ) ,
315 while ( count-- )
316 bitgen ( bits , index_addr , subtree ( 2 , T ) ) ,
317 break ,
318 default :
319 lm_Error
320 ( 1 ,
321 /*MSG*/"internal error: bitgen: bad node"
322 ) ,
323 lm_Fail ( ) ,
324 } /* bitgen */
325
326 static char *
327 gen_bits ( len , T )
328 ushort len ,
329 tree T ,
330 /* allocates, generates, and returns the bit sequence for T */
331 { char *ret ,
332 ushort adex = 0 ,
333 ret = allo_bits ( len ) ,
334 bitgen ( ret , &adex , T ) ,
335 ASSERT(adex==len) ,
336 return ret ,
337 } /* gen_bits */
338
339 static ushort
340 map_drive ( drv )
341 ushort drv ,
342 /* maps constrainer drive types to device structure drive types */
343 { switch ( drv )
344 { case C_hard :
345 return H_DRIVE ,
346 case C_medium :
347 return M_DRIVE ,
348 case C_hard_medium :
349 return HM_DRIVE ,
350 default :
351 lm_Error
352 ( 1 ,
353 /*MSG*/"internal error: bad drive"
354 ) ,
355 return NO_DRIVE ,
356 }
357 } /* map_drive */
358
359 static ushort
360

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hgen.c	DATE 5/23/89	PAGE # 4/14
LINE #		SOURCE TEXT		
361		map_soft_drive (val , w1 , w2 , w3 , w4 , round_up)		
362		{		
363		ushort		
364		val ,		
365		w1 ,		
366		w2 ,		
367		w3 ,		
368		w4 ,		
369		round_up ,		
370		/* maps val into a 4-bit number based on the passed weights */		
371		{ ushort ret ;		
372		ret = 0 ;		
373		if (val >= w4)		
374		{ ret = 8 ;		
375		val -= w4 ;		
376		if (val >= w3)		
377		{ ret = 4 ;		
378		val -= w3 ;		
379		if (val >= w2)		
380		{ ret = 2 ;		
381		val -= w2 ;		
382		if (val >= w1)		
383		{ ret = 1 ;		
384		val -= w1 ;		
385		if (round_up && val > 0 && ret != 15)		
386		++ret ;		
387		return ret ;		
388		} /* map_soft_drive */		
389		static void		
390		edge_fall_filter (bits , len)		
391		{		
392		char *bits ;		
393		ushort len ;		
394		{ ushort i , prev = 0 , byte , mod ;		
395		for (i = 0 ; i < len ; ++i)		
396		{ byte = bits[i] >> 3 ;		
397		mod = i & 7 ;		
398		if (prev)		
399		{ prev = (bits [byte] >> (7 - mod)) & 1 ;		
400		bits [byte] = 1 << (7 - mod) ;		
401		else		
402		{ prev = (bits [byte] >> (7 - mod)) & 1 ;		
403		if (prev)		
404		bits [byte] = 1 << (7 - mod) ;		
405		}		
406		}		
407		} /* edge_fall_filter */		
408		static void		
409		edge_rise_filter (bits , len)		
410		{		
411		char *bits ;		
412		ushort len ;		
413		{ ushort i , prev = 1 , byte , mod ;		
414		for (i = 0 ; i < len ; ++i)		
415		{ byte = bits[i] >> 3 ;		
416		mod = i & 7 ;		
417		if (! prev)		
418		{ prev = (bits [byte] >> (7 - mod)) & 1 ;		
419		bits [byte] = 1 << (7 - mod) ;		
420		else		
421		{ prev = (bits [byte] >> (7 - mod)) & 1 ;		
422		if (! prev)		
423		bits [byte] = 1 << (7 - mod) ;		
424		}		
425		}		
426		} /* edge_rise_filter */		
427		static ushort		
428		seq_len (T)		
429		{		
430		tree T ;		
431		{ ushort ret , i , kids ;		
432		kids = kidcount (T) ;		
433		switch (nameofnode (T))		
434		{ case P_NUM : return strlen (symtostr (textofnode (T))) ;		
435		case M_LIST :		
436		ret = 0 ;		
437		for (i = 1 , i <= kids ; ++i)		
438		ret += seq_len (subtree (i , T)) + (i != 1) ;		
439		break ;		
440		case N_REPEAT :		
441		ret = strlen (symtostr (textofnode (subtree (1 , T)))) ;		
442		ret += seq_len (subtree (2 , T)) * 2 ;		
443		break ;		
444		default :		
445		ln_error (
446		{ 1 ,		
447		"internal error: seq_len: bad node"		
448) ;		
449		ln_fatal () ;		
450		}		
451		}		
452		return ret ;		
453		} /* seq_len */		
454		static void		
455		gen_str (str , ndex , T)		
456		{		
457		char *str ;		
458		ushort *ndex ;		
459		tree T ;		
460		{ ushort i , kids ;		
461		char *s ;		
462		kids = kidcount (T) ;		
463		switch (nameofnode (T))		
464		{ case P_NUM :		
465		s = symtostr (textofnode (T)) ;		
466		(void) strcpy (str + *ndex , s) ;		
467		*ndex += strlen (s) ;		
468		break ;		
469		case N_LIST :		
470		for (i = 1 , i <= kids ; ++i)		
471		{ if (i != 1)		
472		(void) strcpy (str + (*ndex)++ , " , ") ;		
473		gen_str (str , ndex , subtree (i , T)) ;		
474		}		
475		}		
476		break ;		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89
TIME 4:42:19 pm

PAGE #
5/15

```

LINE #          SOURCE TEXT
481      case N_REPEAT :
482      case N_FEEDBACK :
483      a = systostr ( tnode ( subtree ( 1, T ) ) ) ;
484      ( void ) strcpy ( str + "ndex", a ) ;
485      *ndex = strlen ( a ) ;
486      ( void ) strcpy ( str + ("ndex")++, " " ) ;
487      gen_str ( str, ndex, subtree ( 2, T ) ) ;
488      ( void ) strcpy ( str + ("ndex")++, " " ) ;
489      break ;
490      default :
491      lm_Error
492      ( 1,
493      /*MSG*/"internal error: gen_str: bad node"
494      ) ;
495      lm_Ffail ( ) ;
496      }
497      /* gen_str */
498
499      static char *
500      gen_seq_str ( T )
501      {
502      char *seq_str ;
503      ushort len ;
504      len = seq_len ( T ) ;
505      seq_str = lm_malloc ( ++len ) ;
506      text_size += len ;
507      if ( seq_str == NULL )
508      {
509      lm_Error
510      ( 1,
511      /*MSG*/"out of memory on modular (gen_seq_str)"
512      ) ;
513      lm_Ffail ( ) ;
514      }
515      len = 0 ;
516      gen_str ( seq_str, &len, T ) ;
517      return seq_str ;
518      /* gen_seq_str */
519
520      static void
521      generate ( t, dev )
522      struct dev *t ;
523      struct device *dev ;
524      /* generates dev from t */
525      {
526      ushort
527      i ;
528      ndex ;
529      count ;
530      stype ;
531      timings ;
532      iih_set ;
533      iil_set ;
534      last_cyc_set ;
535      tree_bits ;
536      symbol_text ;
537      PIN_SPEC *pin ;
538      struct timing *tim ;
539      struct pin_attr *attr ;
540      text_size = 0 ;
541      dev->device_name = allo_string ( t->dev_name ) ;
542      dev->modular_name =
543      t->mod_name == NULLSYM ?
544      NULL :
545      allo_string ( t->mod_name ) ;
546      dev->timing_cnt =
547      dev->stype_cnt =
548      0 ;
549      dev->extra_data = NULL ;
550      dev->next_device = NULL ;
551      dev->clock_period1 = t->clk_period1 ;
552      dev->clock_period2 = t->clk_period2 == 0 ? -1 : t->clk_period2 ;
553      dev->vdd = t->Vdd * 1000 + 0.5 ;
554      dev->vss = t->Vss * 1000 + 0.5 ;
555      dev->vth = t->Vth * 1000 + 0.5 ;
556      dev->vthl = t->Vthl * 1000 + 0.5 ;
557      dev->vll = t->Vll * 1000 + 0.5 ;
558      dev->vth = t->Vth * 1000 + 0.5 ;
559      dev->vcc = t->Vcc * 1000 + 0.5 ;
560      dev->setup_time =
561      t->def_setup != UNDEFINED ? t->def_setup + 0.5 : -1 ;
562      dev->hold_time =
563      t->def_hold != UNDEFINED ? t->def_hold + 0.5 : -1 ;
564      if ( t->def_sample == UNDEFINED )
565      dev->five_state_sample = 0 ;
566      else
567      dev->five_state_sample = t->def_sample + 0.5 ;
568      dev->two_state_sample = 0 ; /* anything else here? */
569      dev->edge_skew_time = EDGE_SKEW_TIME ;
570      dev->no_settling_time = NO_SETTLING_TIME ;
571      dev->no_settling_time = NO_SETTLING_TIME ;
572      dev->clock_type =
573      t->clk_type == C_internal ?
574      INTERNAL :
575      t->clk_type == C_ext1 ?
576      EXT1 :
577      EXT2 ;
578      dev->device_type = t->dev_type == C_public ? PUBLIC : PRIVATE ;
579      dev->fb_type = t->fb ? FB : FEEDBACK_RISE : FEEDBACK_FALL ;
580      dev->dis_tin_check = t->dis_tin_check ;
581      dev->inh_tin_measure = t->inh_tin_measure ;
582      dev->ultra_fast = t->ultra_fast ;
583      dev->use_default = t->use_default ;
584      dev->default_delay . minimum = t->min_default ;
585      dev->default_delay . typical = t->typ_default ;
586      dev->default_delay . maximum = t->max_default ;
587      dev->report_mis =
588      the->missing_delays == NULLSYM ?
589      the->delays_seen :
590      the->missing_delays == C_on ;
591      dev->report_ioec = the->io_store_changes == C_on ;
592      dev->pin_table = allo_pias ( dev->pin_cnt = t->pin_cnt ) ;
593      for ( i = 0 ; i < t->pin_cnt ; ++i )
594      if ( t->pin [ i ] . number < MAX_PINS )
595      {
596      pin = dev->pin_table + t->pin [ i ] . number ;
597      ASSERT ( pin->direction == NONE ) ;
598      iih_set = iil_set = last_cyc_set = 0 ;
599      text = t->pin [ i ] . alias ;
600      pin->pin_alias = text == NULLSYM ? NULL : allo_string ( text ) ;
601      text = t->pin [ i ] . pkg_name ;

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hgen.c	DATE 5/23/89	PAGE # 6/16
LINE #	SOURCE TEXT			
601	pin -> pin_number = allo_string (text) ;			
602	text = t -> pins [i] . name ;			
603	pin -> pin_name = allo_string (text) ;			
604	pin -> trace_delay = t -> pins [i] . trace_delay ;			
605	switch (nameofnode (g_lookup (text)))			
606	{ case M_IN_PINS :			
607	pin -> direction = IN ;			
608	pin -> is_seq_drive = M_DRIVE ;			
609	pin -> last_cyc_drive = M_DRIVE ;			
610	pin -> h_drive = ALWAYS_ON ;			
611	pin -> hm_drive = DRIVE_ON ;			
612	pin -> s_drive_low =			
613	map_soft_drive (0 , 200 , 500 , 1200 , 3000 , 0) ;			
614	pin -> s_drive_hi =			
615	map_soft_drive (0 , 100 , 500 , 1200 , 2000 , 0) ;			
616	break ;			
617	case M_OUT_PINS :			
618	pin -> direction = OUT ;			
619	pin -> is_seq_drive = NO_DRIVE ;			
620	pin -> last_cyc_drive = NO_DRIVE ;			
621	pin -> h_drive = GATED_ON ;			
622	pin -> hm_drive = DRIVE_OFF ;			
623	pin -> s_drive_low =			
624	map_soft_drive			
625	{ (ushort) (t -> Isl * 1000 + 0.5) ,			
626	200 ,			
627	500 ,			
628	1200 ,			
629	3000 ,			
630	0			
631	}			
632	pin -> s_drive_hi =			
633	map_soft_drive (0 , 100 , 500 , 1200 , 2000 , 0) ;			
634	break ;			
635	case M_IO_PINS :			
636	pin -> direction = IO ;			
637	pin -> is_seq_drive = t -> ultra_fast ? HM_DRIVE : M_DRIVE ;			
638	pin -> last_cyc_drive = pin -> is_seq_drive ;			
639	pin -> h_drive = GATED_ON ;			
640	pin -> hm_drive = DRIVE_ON ;			
641	pin -> s_drive_low =			
642	map_soft_drive			
643	{ (ushort) (t -> Isl * 1000 + 0.5) ,			
644	200 ,			
645	500 ,			
646	1200 ,			
647	3000 ,			
648	0			
649	}			
650	pin -> s_drive_hi =			
651	map_soft_drive			
652	{ (ushort) (t -> Ish * 1000 + 0.5) ,			
653	100 ,			
654	500 ,			
655	1200 ,			
656	3000 ,			
657	0			
658	}			
659	break ;			
660	case M_POWER_PINS :			
661	pin -> direction = POWER ;			
662	continue ;			
663	case M_GROUND_PINS :			
664	pin -> direction = GROUND ;			
665	continue ;			
666	case M_NC_PINS :			
667	pin -> direction = NC ;			
668	continue ;			
669	default :			
670	ASSERT(!--zero);			
671	break ;			
672	{			
673	pin -> clk_edge1 = 0 ;			
674	pin -> clk_edge2 = 0 ;			
675	pin -> clk_format = DMZ ;			
676	pin -> pin_class = DATA ;			
677	pin -> feeds_back = 0 ;			
678	pin -> kept_alive = 0 ;			
679	pin -> pulled_up = 0 ;			
680	pin -> pulled_down = 0 ;			
681	for (j = 0 , j < t -> pins [i] . num_attr , ++j)			
682	{ attr = t -> pins [i] . attr + j ;			
683	switch (attr -> use_flag)			
684	{ case ID_ATTR :			
685	switch (attr -> attr . id)			
686	{ case C_eval :			
687	pin -> pin_class = EVAL ;			
688	break ;			
689	case C_store :			
690	pin -> pin_class = STORE ;			
691	pin -> clk_format = DMZ ;			
692	break ;			
693	case C_edge_rise :			
694	pin -> pin_class = STORE ;			
695	pin -> clk_format =			
696	pin -> direction == IO ? DMZ : R1 ;			
697	break ;			
698	case C_edge_fall :			
699	pin -> pin_class = DMZ ;			
700	pin -> clk_format =			
701	pin -> direction == IO ? DMZ : R0 ;			
702	break ;			
703	case C_feedback :			
704	pin -> feeds_back = 1 ;			
705	break ;			
706	case C_keeplive :			
707	pin -> kept_alive = 1 ;			
708	break ;			
709	case C_pull_up :			
710	pin -> pulled_up = 1 ;			
711	if (! lib_set)			
712	pin -> s_drive_hi =			
713	map_soft_drive			
714	{ (ushort)			
715	{ 2000 * (t -> Vcc / 5) + 0.5			
716	}			
717	100 ,			
718	500 ,			
719	1200 ,			
720	2000 ,			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE	5/23/89	PAGE #
TIME	4:42:19 pm	7/17

```

LINE #      SOURCE TEXT
721          0
722          if ( ! iil_set )
723              pin -> s_drive_low =
724                  map_soft_drive
725                      ( ( ushort )
726                          ( 2000 * ( t -> Vcc / 5 ) + 0.5
727                              ,
728                              ,
729                              ,
730                              ,
731                              ,
732                              ,
733                              ,
734                          )
735                      )
736              break ;
737          case C_pull_down :
738              pin -> pulled_down = 1 ;
739              if ( ! iih_set )
740                  pin -> s_drive_hi =
741                      map_soft_drive
742                          ( ( ushort )
743                              ( 2000 * ( t -> Vcc / 5 ) + 0.5
744                                  ,
745                                  ,
746                                  ,
747                                  ,
748                                  ,
749                                  ,
750                              )
751                          )
752              if ( ! iil_set )
753                  pin -> s_drive_low =
754                      map_soft_drive
755                          ( ( ushort )
756                              ( 2000 * ( t -> Vcc / 5 ) + 0.5
757                                  ,
758                                  ,
759                                  ,
760                                  ,
761                                  ,
762                                  ,
763                              )
764                          )
765              break ;
766          default :
767              is_herror
768              ( 1
769              )
770              break ;
771          break ;
772          case CURRENT_SPEC :
773              switch ( attr -> attr . cur_spec . cur )
774              {
775                  case C_iil :
776                      iil_set = 1 ;
777                      pin -> s_drive_low =
778                          map_soft_drive
779                              ( ( ushort )
780                                  ( attr -> attr . cur_spec . val * 1900 + 0.5
781                                      ,
782                                      ,
783                                      ,
784                                      ,
785                                      ,
786                                      ,
787                                  )
788                              )
789                      break ;
790                  case C_iia :
791                      iia_set = 1 ;
792                      pin -> s_drive_hi =
793                          map_soft_drive
794                              ( ( ushort )
795                                  ( attr -> attr . cur_spec . val * 1900 + 0.5
796                                      ,
797                                      ,
798                                      ,
799                                      ,
800                                      ,
801                                      ,
802                                  )
803                              )
804                      break ;
805                  case C_iol :
806                      break ;
807                  case C_ion :
808                      break ;
809                  case C_pull_up :
810                      pin -> pulled_up = 1 ;
811                      if ( ! iih_set )
812                          pin -> s_drive_hi =
813                              map_soft_drive
814                                  ( ( ushort )
815                                      ( 2000 *
816                                          ( t -> Vcc /
817                                              ( attr -> attr . cur_spec . val /
818                                                  1000.0
819                                              )
820                                          ) + 0.5
821                                      ,
822                                      ,
823                                      ,
824                                      ,
825                                      ,
826                                      ,
827                                      ,
828                                      ,
829                                      ,
830                                      ,
831                                      ,
832                                      ,
833                                      ,
834                                      ,
835                                      ,
836                                      ,
837                                      ,
838                                      ,
839                                      ,
840                                      ,
841                                      ,
842                                      ,
843                                      ,
844                                      ,
845                                      ,
846                                      ,
847                                      ,
848                                      ,
849                                      ,
850                                      ,
851                                      ,
852                                      ,
853                                      ,
854                                      ,
855                                      ,
856                                      ,
857                                      ,
858                                      ,
859                                      ,
860                                      ,
861                                      ,
862                                      ,
863                                      ,
864                                      ,
865                                      ,
866                                      ,
867                                      ,
868                                      ,
869                                      ,
870                                      ,
871                                      ,
872                                      ,
873                                      ,
874                                      ,
875                                      ,
876                                      ,
877                                      ,
878                                      ,
879                                      ,
880                                      ,
881                                      ,
882                                      ,
883                                      ,
884                                      ,
885                                      ,
886                                      ,
887                                      ,
888                                      ,
889                                      ,
890                                      ,
891                                      ,
892                                      ,
893                                      ,
894                                      ,
895                                      ,
896                                      ,
897                                      ,
898                                      ,
899                                      ,
900                                      ,
901                                      ,
902                                      ,
903                                      ,
904                                      ,
905                                      ,
906                                      ,
907                                      ,
908                                      ,
909                                      ,
910                                      ,
911                                      ,
912                                      ,
913                                      ,
914                                      ,
915                                      ,
916                                      ,
917                                      ,
918                                      ,
919                                      ,
920                                      ,
921                                      ,
922                                      ,
923                                      ,
924                                      ,
925                                      ,
926                                      ,
927                                      ,
928                                      ,
929                                      ,
930                                      ,
931                                      ,
932                                      ,
933                                      ,
934                                      ,
935                                      ,
936                                      ,
937                                      ,
938                                      ,
939                                      ,
940                                      ,
941                                      ,
942                                      ,
943                                      ,
944                                      ,
945                                      ,
946                                      ,
947                                      ,
948                                      ,
949                                      ,
950                                      ,
951                                      ,
952                                      ,
953                                      ,
954                                      ,
955                                      ,
956                                      ,
957                                      ,
958                                      ,
959                                      ,
960                                      ,
961                                      ,
962                                      ,
963                                      ,
964                                      ,
965                                      ,
966                                      ,
967                                      ,
968                                      ,
969                                      ,
970                                      ,
971                                      ,
972                                      ,
973                                      ,
974                                      ,
975                                      ,
976                                      ,
977                                      ,
978                                      ,
979                                      ,
980                                      ,
981                                      ,
982                                      ,
983                                      ,
984                                      ,
985                                      ,
986                                      ,
987                                      ,
988                                      ,
989                                      ,
990                                      ,
991                                      ,
992                                      ,
993                                      ,
994                                      ,
995                                      ,
996                                      ,
997                                      ,
998                                      ,
999                                      ,
1000                                     )

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89
TIME 4:42:19 pm

PAGE #
8/18

```

LINE # SOURCE TEXT
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960

```

```

1
if ( pin -> a_drive_low == 0 )
pin -> a_drive_low = 1 ;
break ;
case C_pull_down :
pin -> pulled_down = 1 ;
if ( ! iih_set )
{ pin -> a_drive_hi =
map_soft_drive
( ( ushort )
( 2000 *
( t -> Vcc /
( attr -> attr . cur_spec . val /
1000.0
) ) + 0.5
) ;
100 ;
500 ;
1200 ;
2000 ;
1 ;
} ;
if ( pin -> a_drive_hi == 0 )
pin -> a_drive_hi = 1 ;
if ( ! iil_set )
{ pin -> a_drive_low =
map_soft_drive
( ( ushort )
( 2000 *
( t -> Vcc /
( attr -> attr . cur_spec . val /
1000.0
) ) + 0.5
) ;
200 ;
500 ;
1200 ;
2000 ;
1 ;
} ;
if ( pin -> a_drive_low == 0 )
pin -> a_drive_low = 1 ;
break ;
default :
lm_Error
( 1 ,
/*LMSC*/"internal error: bad cur_spec"
) ;
break ;
break ;
case CTRL_SPEC :
switch ( attr -> attr . cyc_spec . seq )
{ case C_ia_sequence :
pin -> ia_seq_drive =
map_drive ( attr -> attr . cyc_spec . drv ) ;
break ;
case C_last_cycle :
pin -> last_cyc_drive =
map_drive ( attr -> attr . cyc_spec . drv ) ;
last_cyc_set = 1 ;
break ;
default :
lm_Error
( 1 ,
/*LMSC*/"internal error: bad seq_spec"
) ;
break ;
} ;
break ;
break ;
default :
lm_Error
( 1 ,
/*LMSC*/"internal error: bad use_flag"
) ;
break ;
} ;
/*h_drive*/
/*hm_drive*/
tim = t -> pins [ i ] . timing_list ;
while ( tim != NULL )
{ ++ dev -> timing_cnt ;
if ( ! is_dup_stym ( t , tim ) )
++ dev -> stym_cnt ;
tim = tim -> next ;
}
if
( pin -> direction == IO &&
pin -> pin_class == STORE &&
! last_cyc_set
)
pin -> t_cyc_drive = NO_DRIVE ;
dev -> timing_table = allo_timings ( dev -> timing_cnt ) ;
dev -> stym_table = allo_styms ( dev -> stym_cnt ) ;
timings =
styms =
0 ;
for ( i = 0 ; i < t -> pin_cnt ; ++i )
if ( t -> pins [ i ] . number < MAX_PINS )
{ pin = dev -> pin_table + t -> pins [ i ] . number ;
ASSERT ( pin -> direction != NONE ) ;
if
( pin -> direction == POWER ||
pin -> direction == GROUND ||
pin -> direction == NC
)
continue ;
count = 0 ;
pin -> delay_table = dev -> timing_table + timings ;
tim = t -> pins [ i ] . timing_list ;
while ( tim != NULL )
{ styms += add_timing ( dev , t , tim , timings , styms ) ;
}
}

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE	5/23/89	PAGE #
TIME	4:42:19 pm	9/19

```

LINE #          SOURCE TEXT
961      ++count .
962      ++timings .
963      tim = tim -> next .
964      }
965      pin -> delay_cnt = count .
966      }
967      ASSERT(timings==dev->timings_cnt);
968      ASSERT(mtyms==dev->mtyms_cnt);
969      dev -> seq_table = allo_seqs ( dev -> seq_cnt = t -> num_seqs );
970      dev -> pre_seq_len = t -> pre_seq_length ;
971      dev -> fb_seq_len = t -> fb_seq_length ;
972      dev -> post_seq_len = t -> post_seq_length ;
973      for ( i = 0 ; i < t -> num_seqs ; ++i )
974      { dev -> seq_table [ i ] . pin_number =
975        t -> pins
976        [ pin_name_of ( t -> sequences [ i ] . pin_name )
977        ] . number ;
978        fb = t -> sequences [ i ] . fb_id ;
979        bits = t -> sequences [ i ] . bits ;
980        if ( fb == 0 )
981        { dev -> seq_table [ i ] . pre_bits =
982          gen_bits ( dev -> pre_seq_len , bits ) ;
983          dev -> seq_table [ i ] . fb_bits = NULL ;
984          dev -> seq_table [ i ] . post_bits = NULL ;
985        }
986        else if ( fb == 1 )
987        { dev -> seq_table [ i ] . pre_bits = NULL ;
988          dev -> seq_table [ i ] . fb_bits =
989            gen_bits
990              ( dev -> fb_seq_len ,
991                subtree ( 2 , subtree ( 1 , bits ) ) )
992          ;
993          if ( kidcount ( bits ) > 1 )
994          { dev -> seq_table [ i ] . post_bits =
995            allo_bits ( dev -> post_seq_len ) ;
996            ndex = 0 ;
997            for ( count = 2 ; count <= kidcount ( bits ) ; ++count )
998              bitgen
999                ( dev -> seq_table [ i ] . post_bits ,
1000                  & ndex ,
1001                  subtree ( count , bits ) )
1002              ;
1003            ASSERT(ndex==dev->post_seq_len);
1004          }
1005          else
1006            dev -> seq_table [ i ] . post_bits = NULL ;
1007        }
1008        else
1009        { dev -> seq_table [ i ] . pre_bits =
1010          allo_bits ( dev -> pre_seq_len ) ;
1011          ndex = 0 ;
1012          for ( count = 1 ; count < fb ; ++count )
1013            bitgen
1014              ( dev -> seq_table [ i ] . pre_bits ,
1015                & ndex ,
1016                subtree ( count , bits ) )
1017            ;
1018            ASSERT(ndex==dev->pre_seq_len);
1019            dev -> seq_table [ i ] . fb_bits =
1020              gen_bits
1021                ( dev -> fb_seq_len ,
1022                  subtree ( 2 , subtree ( fb , bits ) ) )
1023              ;
1024            if ( kidcount ( bits ) > fb )
1025            { dev -> seq_table [ i ] . post_bits =
1026              allo_bits ( dev -> post_seq_len ) ;
1027              ndex = 0 ;
1028              for ( count = fb + 1 ; count <= kidcount ( bits ) ; ++count )
1029                bitgen
1030                  ( dev -> seq_table [ i ] . post_bits ,
1031                    & ndex ,
1032                    subtree ( count , bits ) )
1033                ;
1034              ASSERT(ndex==dev->post_seq_len);
1035            }
1036            else
1037              dev -> seq_table [ i ] . post_bits = NULL ;
1038          }
1039          dev -> seq_table [ i ] . seq_str =
1040            gen_seq_str ( t -> sequences [ i ] . bits ) ;
1041          if
1042            { dev -> pin_table
1043              [ dev -> seq_table [ i ] . pin_number
1044              ] . clk_format == R0
1045            }
1046            { edge_fall_filter
1047              ( dev -> seq_table [ i ] . pre_bits ,
1048                dev -> pre_seq_len
1049              ) ;
1050              edge_fall_filter
1051                ( dev -> seq_table [ i ] . fb_bits ,
1052                  dev -> fb_seq_len
1053                ) ;
1054              edge_fall_filter
1055                ( dev -> seq_table [ i ] . post_bits ,
1056                  dev -> post_seq_len
1057                ) ;
1058            }
1059          else if
1060            { dev -> pin_table
1061              [ dev -> seq_table [ i ] . pin_number
1062              ] . clk_format == R1
1063            }
1064            { edge_rise_filter
1065              ( dev -> seq_table [ i ] . pre_bits ,
1066                dev -> pre_seq_len
1067              ) ;
1068              edge_rise_filter
1069                ( dev -> seq_table [ i ] . fb_bits ,
1070                  dev -> fb_seq_len
1071                ) ;
1072              edge_rise_filter
1073                ( dev -> seq_table [ i ] . post_bits ,
1074                  dev -> post_seq_len
1075                ) ;
1076            }
1077          }
1078          } /* generate */
1079
1080 # ifdef WRITE_DEVICE

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89
TIME 4:42:19 pm

PAGE #
10/20

```

1081 static void
1082 write_str ( str , F )
1083 char *str ;
1084 FILE *F ;
1085 /* writes passed string to passed file */
1086 { ( void ) fwrite ( str , strlen ( str ) , 1 , F ) ;
1087 } /* write_str */
1088
1089 ushort
1090 write_device ( F , dev )
1091 FILE *F ;
1092 struct device *dev ;
1093 /* writes the passed device to the passed file */
1094 { ushort
1095   i ,
1096   len1 ,
1097   len2 ,
1098   len3 ;
1099   unsigned long
1100   text_offset ;
1101   dev_size = dev-> pin_cnt * sizeof ( PIN_SPEC ) +
1102   pin_size = dev-> seq_cnt * sizeof ( SEQ_SPEC ) +
1103   seq_size = dev-> timing_cnt * sizeof ( TIMING_SPEC ) +
1104   time_size = dev-> mtyx_cnt * sizeof ( MIN_TTP_MAX ) ;
1105   char
1106   *save1 ,
1107   *save2 ,
1108   *save3 ,
1109   *save4 ,
1110   *save5 ,
1111   *save6 ,
1112   *save7 ;
1113   PIN_SPEC *save1 ;
1114   SEQ_SPEC *save2 ;
1115   TIMING_SPEC *save3 ;
1116   MIN_TTP_MAX *save4 ;
1117   text_offset =
1118   ( dev_size = sizeof ( DEVICE_SPEC ) ) +
1119   ( pin_size = dev-> pin_cnt * sizeof ( PIN_SPEC ) ) +
1120   ( seq_size = dev-> seq_cnt * sizeof ( SEQ_SPEC ) ) +
1121   ( time_size = dev-> timing_cnt * sizeof ( TIMING_SPEC ) ) +
1122   ( dev-> mtyx_cnt * sizeof ( MIN_TTP_MAX ) ) ;
1123   offset = text_offset + text_size ;
1124   ( void ) fwrite ( ( char * ) &offset , sizeof ( offset ) , 1 , F ) ;
1125   /*(void)printf(stderr,"offset=%d\n",offset);*/
1126   save1 = dev-> device_name ;
1127   dev-> device_name = ( char * ) text_offset ;
1128   text_offset += strlen ( save1 ) + 1 ;
1129   save2 = dev-> modeler_name ;
1130   if ( save2 != NULL )
1131   { dev-> modeler_name = ( char * ) text_offset ;
1132     text_offset += strlen ( save2 ) + 1 ;
1133   }
1134   save1 = dev-> pin_table ;
1135   dev-> pin_table = ( PIN_SPEC * ) ( offset = dev_size ) ;
1136   save2 = dev-> seq_table ;
1137   dev-> seq_table = ( SEQ_SPEC * ) ( offset += pin_size ) ;
1138   save3 = dev-> timing_table ;
1139   dev-> timing_table = ( TIMING_SPEC * ) ( offset += seq_size ) ;
1140   save4 = dev-> mtyx_table ;
1141   dev-> mtyx_table = ( MIN_TTP_MAX * ) ( offset += time_size ) ;
1142   ( void ) fwrite ( ( char * ) dev , sizeof ( *dev ) , 1 , F ) ;
1143   dev-> device_name = save1 ;
1144   dev-> modeler_name = save2 ;
1145   dev-> pin_table = save3 ;
1146   dev-> seq_table = save4 ;
1147   dev-> timing_table = save5 ;
1148   dev-> mtyx_table = save6 ;
1149   /*(void)printf(stderr,"about to write pins\n");*/
1150   offset = time_size ;
1151   for ( i = 0 ; i < dev-> pin_cnt ; ++i )
1152   { save1 = dev-> pin_table + i ;
1153     if ( save1-> direction == NONE )
1154     { ( void ) fwrite ( ( char * ) save1 , sizeof ( *save1 ) , 1 , F ) ;
1155     }
1156     else
1157     { save1 = save1-> pin_name ;
1158       save2 = save1-> pin_number ;
1159       save3 = save1-> pin_alias ;
1160       save4 = save1-> pin_name + ( char * ) text_offset ;
1161       text_offset += strlen ( save4 ) + 1 ;
1162       save5 = save1-> pin_number + ( char * ) text_offset ;
1163       text_offset += strlen ( save5 ) + 1 ;
1164       save6 = save1-> pin_alias + ( char * ) text_offset ;
1165       text_offset += strlen ( save6 ) + 1 ;
1166     }
1167     if ( save1-> direction != POWER &&
1168         save1-> direction != GROUND &&
1169         save1-> direction != NC )
1170     { save1 = save1-> delay_table ;
1171       save2 = save1-> delay_table =
1172       ( TIMING_SPEC * ) offset + ( save1-> dev-> timing_table ) ;
1173       ( void ) fwrite
1174       ( ( char * ) save1 , sizeof ( *save1 ) , 1 , F ) ;
1175       save1-> delay_table = save2 ;
1176     }
1177     else
1178     { ( void ) fwrite ( ( char * ) save1 , sizeof ( *save1 ) , 1 , F ) ;
1179     }
1180     save1-> pin_name = save4 ;
1181     save2-> pin_number = save5 ;
1182     save3-> pin_alias = save6 ;
1183   }
1184   /*(void)printf(stderr,"about to write seqs\n");*/
1185   len1 = ( ( dev-> pre_seq_len - 1 ) >> 3 ) + 1 ;
1186   len2 = ( ( dev-> fb_seq_len - 1 ) >> 3 ) + 1 ;
1187   len3 = ( ( dev-> post_seq_len - 1 ) >> 3 ) + 1 ;
1188   for ( i = 0 ; i < dev-> seq_cnt ; ++i )
1189   { save1 = dev-> seq_table + i ;
1190     if ( len1 )
1191     { save1 = save1-> pre_bits ;
1192       save2 = save1-> pre_bits + ( char * ) text_offset ;
1193       text_offset += len1 ;
1194     }
1195     if ( len2 )
1196     { save1 = save1-> fb_bits ;
1197       save2 = save1-> fb_bits + ( char * ) text_offset ;
1198       text_offset += len2 ;
1199     }
1200   }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hgen.c

DATE 5/23/89
TIME 4:42:19 pm

PAGE #
11/21

```

1201     if ( len1 )
1202     {
1203         saved -> post_bits = ( char * ) text_offset ;
1204         text_offset += len1 ;
1205     }
1206     save0 = saved -> seq_str ;
1207     save4 -> seq_str = ( char * ) text_offset ;
1208     text_offset += strlen ( save0 ) + 1 ;
1209     ( void ) fwrite ( ( char * ) save4 , sizeof ( 'save4' ) , 1 , F ) ;
1210     if ( len1 )
1211         saved -> pre_bits = save1 ;
1212     if ( len2 )
1213         saved -> fb_bits = save2 ;
1214     if ( len3 )
1215         saved -> post_bits = save3 ;
1216     save4 -> seq_str = save0 ;
1217 }
1218 /*(void)fprintf(stderr,"about to write timing\n");*/
1219 ( void ) fwrite ( dev -> timing_table ,
1220     sizeof ( dev -> timing_table [ 0 ] ) ,
1221     ( int ) dev -> timing_cnt ,
1222     F ) ;
1223 /*(void)fprintf(stderr,"about to write stym\n");*/
1224 ( void ) fwrite ( dev -> stym_table ,
1225     sizeof ( dev -> stym_table [ 0 ] ) ,
1226     ( int ) dev -> stym_cnt ,
1227     F ) ;
1228 /*(void)fprintf(stderr,"about to write text\n");*/
1229 write_str ( dev -> device_name , F ) ;
1230 if ( dev -> modeler_name != NULL )
1231     write_str ( dev -> modeler_name , F ) ;
1232 for ( i = 0 ; i < dev -> pin_cnt ; ++i )
1233     if ( dev -> pin_table [ i ] . direction != NONE )
1234     {
1235         write_str ( dev -> pin_table [ i ] . pin_name , F ) ;
1236         write_str ( dev -> pin_table [ i ] . pin_number , F ) ;
1237         if ( dev -> pin_table [ i ] . pin_alias != NULL )
1238             write_str ( dev -> pin_table [ i ] . pin_alias , F ) ;
1239     }
1240 for ( i = 0 ; i < dev -> seq_cnt ; ++i )
1241     if ( dev -> pre_seq_len )
1242     {
1243         ( void ) fwrite ( dev -> seq_table [ i ] . pre_bits ,
1244             ( int ) len1 ,
1245             1 ,
1246             F ) ;
1247     }
1248     if ( dev -> fb_seq_len )
1249     {
1250         ( void ) fwrite ( dev -> seq_table [ i ] . fb_bits ,
1251             ( int ) len2 ,
1252             1 ,
1253             F ) ;
1254     }
1255     if ( dev -> post_seq_len )
1256     {
1257         ( void ) fwrite ( dev -> seq_table [ i ] . post_bits ,
1258             ( int ) len3 ,
1259             1 ,
1260             F ) ;
1261     }
1262     write_str ( dev -> seq_table [ i ] . seq_str , F ) ;
1263 }
1264 /*(void)fprintf(stderr,"about to close\n");*/
1265 return fclose ( F ) == 0 ;
1266 } /* write_device */
1267 #endif WRITE_DEVICE
1268
1269 void
1270 free_dev ( dev )
1271 struct dev *dev ;
1272 {
1273     ushort i ;
1274     struct pins *pin ;
1275     struct pin_attr *attr , *prev_attr ;
1276     struct timing *tim ;
1277     prev_attr = NULL ;
1278     if ( dev == NULL )
1279         return ;
1280     for ( i = 0 ; i < dev -> pin_cnt ; ++i )
1281     {
1282         pin = dev -> pins + i ;
1283         attr = pin -> attr ;
1284         if ( attr != prev_attr )
1285             ( void ) lm_free ( ( char * ) attr ) ;
1286         prev_attr = attr ;
1287     }
1288     while ( pin -> timing_list != NULL )
1289     {
1290         tim = pin -> timing_list -> next ;
1291         ( void ) lm_free ( ( char * ) pin -> timing_list ) ;
1292         pin -> timing_list = tim ;
1293     }
1294     ( void ) lm_free ( ( char * ) dev -> pins ) ;
1295     if ( dev -> sequences != NULL )
1296         ( void ) lm_free ( ( char * ) dev -> sequences ) ;
1297     ( void ) lm_free ( ( char * ) dev ) ;
1298 } /* free_dev */
1299
1300 void
1301 init_the_device ( )
1302 {
1303     device = NULL ;
1304 } /* init_the_device */
1305
1306 void
1307 free_the_device ( )
1308 {
1309     free_dev ( device ) ;
1310 } /* free_the_device */
1311
1312 struct device *
1313 gen ( t )
1314 struct dev *t ;
1315 /* returns pointer to device structure generated from passed device */
1316 {
1317     device = allo_device ( ) ;
1318     generate ( t , device ) ;
1319     free_dev ( t ) ;
1320     return ( device ) ;
1321 }

```

701

5,353,243

702

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hgen.c	DATE 5/23/89	PAGE # 12/22
		TIME 4:42:19 pm		
LINE #	SOURCE TEXT			
1321	return device /			
1322	1 /* gen */			
1323				
1324	/* end of Generator for ENCL Processor */			
1325				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE 5/23/89
TIME 4:42:21 pm

PAGE #
1/23

```

LINE #          SOURCE TEXT
1  /* SCCS ID: Hparse.c rev 3.1, 4/24/89 at 08:00:42 */
2  /* recursive descent parser generated from "HMBL.g" */
3  /*
4  */
5  #include <strings.h>
6  #include "strng.h"
7  #include "tree.h"
8
9  extern void
10  scan ( ) ;
11  syntax_error ( ) ;
12
13  extern symbol
14  tok_name ;
15  tok_text ;
16  #include "Hparse.h"
17
18  #define Parse_Statement_list Q_1
19  #define Parse_Statement Q_2
20  #define Parse_Name Q_3
21  #define Parse_Opt_id Q_4
22  #define Parse_Unit Q_5
23  #define Parse_Opt_dash_unit Q_6
24  #define Parse_Opt_tech_spec_list Q_7
25  #define Parse_Init_seq_list Q_8
26  #define Parse_Pin_corr_list Q_9
27  #define Parse_From_list Q_10
28  #define Parse_Timing Q_11
29  #define Parse_Package_mapping_list Q_12
30  #define Parse_Adapter_mapping_list Q_13
31  #define Parse_Pin_name_mapping_list Q_14
32  #define Parse_Tech_spec_list Q_15
33  #define Parse_Tech_spec Q_16
34  #define Parse_Signed_num Q_17
35  #define Parse_Init_seq Q_18
36  #define Parse_Pin_name_list Q_19
37  #define Parse_Init_list Q_20
38  #define Parse_Init_value_list Q_21
39  #define Parse_Init_value Q_22
40  #define Parse_Pin_corr Q_23
41  #define Parse_Names Q_24
42  #define Parse_Opt_pin_attr_list Q_25
43  #define Parse_Pin_name Q_26
44  #define Parse_Subscript Q_27
45  #define Parse_Opt_colon_sum Q_28
46  #define Parse_Name_list Q_29
47  #define Parse_Pin_attr_list Q_30
48  #define Parse_Pin_attr Q_31
49  #define Parse_From Q_32
50  #define Parse_Pin_state Q_33
51  #define Parse_To_list Q_34
52  #define Parse_To Q_35
53  #define Parse_Spec Q_36
54  #define Parse_Package_mapping Q_37
55  #define Parse_Adapter_mapping Q_38
56  #define Parse_Pin_name_mapping Q_39
57  #define Parse_Pin_names Q_40
58
59  static unsigned short
60  Parse_Statement_list ( ) ;
61  Parse_Statement ( ) ;
62  Parse_Name ( ) ;
63  Parse_Opt_id ( ) ;
64  Parse_Unit ( ) ;
65  Parse_Opt_dash_unit ( ) ;
66  Parse_Opt_tech_spec_list ( ) ;
67  Parse_Init_seq_list ( ) ;
68  Parse_Pin_corr_list ( ) ;
69  Parse_From_list ( ) ;
70  Parse_Timing ( ) ;
71  Parse_Package_mapping_list ( ) ;
72  Parse_Adapter_mapping_list ( ) ;
73  Parse_Pin_name_mapping_list ( ) ;
74  Parse_Tech_spec_list ( ) ;
75  Parse_Tech_spec ( ) ;
76  Parse_Signed_num ( ) ;
77  Parse_Init_seq ( ) ;
78  Parse_Pin_name_list ( ) ;
79  Parse_Init_list ( ) ;
80  Parse_Init_value_list ( ) ;
81  Parse_Init_value ( ) ;
82  Parse_Pin_corr ( ) ;
83  Parse_Names ( ) ;
84  Parse_Opt_pin_attr_list ( ) ;
85  Parse_Pin_name ( ) ;
86  Parse_Subscript ( ) ;
87  Parse_Opt_colon_sum ( ) ;
88  Parse_Name_list ( ) ;
89  Parse_Pin_attr_list ( ) ;
90  Parse_Pin_attr ( ) ;
91  Parse_From ( ) ;
92  Parse_Pin_state ( ) ;
93  Parse_To_list ( ) ;
94  Parse_To ( ) ;
95  Parse_Spec ( ) ;
96  Parse_Package_mapping ( ) ;
97  Parse_Adapter_mapping ( ) ;
98  Parse_Pin_name_mapping ( ) ;
99  Parse_Pin_names ( ) ;
100
101  void
102  pinit ( ) ;
103  {
104  ( void ) strtosym ( "END_OF_FILE" ) ;
105  ( void ) strtosym ( "ID" ) ;
106  ( void ) strtosym ( "NOR" ) ;
107  ( void ) strtosym ( "STX" ) ;
108  ( void ) strtosym ( "NAME" ) ;
109  ( void ) strtosym ( "FAST" ) ;
110  ( void ) strtosym ( "USAGE" ) ;
111  ( void ) strtosym ( "REPORT" ) ;
112  ( void ) strtosym ( "MODELER" ) ;
113  ( void ) strtosym ( "CLOCK_TYPE" ) ;
114  ( void ) strtosym ( "DIS_TIM_CHECK" ) ;
115  ( void ) strtosym ( "INS_TIM_MEASURE" ) ;
116  ( void ) strtosym ( "SOLD_TIME" ) ;
117  ( void ) strtosym ( "SETUP_TIME" ) ;
118  ( void ) strtosym ( "SAMPLE_TIME" ) ;
119
120

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hparse.c	DATE 5/23/89 TIME 4:42:21 pm	PAGE # 2/24
LINE #	SOURCE TEXT			
121	void strtosys ("SPEED") ;			
122	void strtosys ("TECHNOLOGY") ;			
123	void strtosys ("INITIALIZE") ;			
124	void strtosys ("IN_PINS") ;			
125	void strtosys ("IO_PINS") ;			
126	void strtosys ("NC_PINS") ;			
127	void strtosys ("OUT_PINS") ;			
128	void strtosys ("POWER_PINS") ;			
129	void strtosys ("GROUND_PINS") ;			
130	void strtosys ("DELAY") ;			
131	void strtosys ("DEFAULT_DELAY") ;			
132	void strtosys ("PACKAGE") ;			
133	void strtosys ("ADAPTER") ;			
134	void strtosys ("PIN_MAP") ;			
135	void strtosys ("UNIT") ;			
136	void strtosys ("TECH_SPEC") ;			
137	void strtosys ("NEGATIVE") ;			
138	void strtosys ("INIT_SEQ") ;			
139	void strtosys ("LIST") ;			
140	void strtosys ("HEPLOT") ;			
141	void strtosys ("FEEDBACK") ;			
142	void strtosys ("PIN CORR") ;			
143	void strtosys ("PIN NAME") ;			
144	void strtosys ("NAMES") ;			
145	void strtosys ("ATTR") ;			
146	void strtosys ("FROM") ;			
147	void strtosys ("TO") ;			
148	void strtosys ("PIN_STATE") ;			
149	void strtosys ("TIMING") ;			
150	void strtosys ("SPEC") ;			
151	void strtosys ("MAPPING") ;			
152	void strtosys ("device_name") ;			
153	void strtosys ("ultra_fast") ;			
154	void strtosys ("device_usage") ;			
155	void strtosys ("report") ;			
156	void strtosys ("modeler_name") ;			
157	void strtosys ("clock_type") ;			
158	void strtosys ("disable_timing_checking") ;			
159	void strtosys ("inhibit_timing_measurement") ;			
160	void strtosys ("device_hold_time") ;			
161	void strtosys ("device_setup_time") ;			
162	void strtosys ("device_sample_time") ;			
163	void strtosys ("device_speed") ;			
164	void strtosys ("technology") ;			
165	void strtosys ("initialize") ;			
166	void strtosys ("in_pin") ;			
167	void strtosys ("io_pin") ;			
168	void strtosys ("nc_pin") ;			
169	void strtosys ("out_pin") ;			
170	void strtosys ("power_pin") ;			
171	void strtosys ("ground_pin") ;			
172	void strtosys ("delay") ;			
173	void strtosys ("default_delay") ;			
174	void strtosys ("package_mapping") ;			
175	void strtosys ("adapter_mapping") ;			
176	void strtosys ("pin_name_mapping") ;			
177	void strtosys ("--") ;			
178	void strtosys ("-") ;			
179	void strtosys ("(") ;			
180	void strtosys (")") ;			
181	void strtosys (".") ;			
182	void strtosys ("[") ;			
183	void strtosys ("]") ;			
184	void strtosys (".") ;			
185	void strtosys ("from") ;			
186	void strtosys ("to") ;			
187	/* pinit */			
188				
189	tree			
190	parse ()			
191	{ scan () , bldnostrm (END_OF_FILE , Parse_Statement_list ()) ;			
192	if (tok_name != END_OF_FILE) syntax_error (1) ;			
193	return popout () ;			
194	/* parse */			
195				
196	static unsigned short			
197	Parse_Statement_list ()			
198	{ unsigned short kids = 0 ;			
199	kids == Parse_Statement () ;			
200	while			
201	{ tok_name == T_device_name			
202	tok_name == T_ultra_fast			
203	tok_name == T_device_usage			
204	tok_name == T_report			
205	tok_name == T_modeler_name			
206	tok_name == T_clock_type			
207	tok_name == T_disable_timing_checking			
208	tok_name == T_inhibit_timing_measurement			
209	tok_name == T_device_hold_time			
210	tok_name == T_device_setup_time			
211	tok_name == T_device_sample_time			
212	tok_name == T_device_speed			
213	tok_name == T_technology			
214	tok_name == T_initialize			
215	tok_name == T_in_pin			
216	tok_name == T_io_pin			
217	tok_name == T_nc_pin			
218	tok_name == T_out_pin			
219	tok_name == T_power_pin			
220	tok_name == T_ground_pin			
221	tok_name == T_delay			
222	tok_name == T_default_delay			
223	tok_name == T_package_mapping			
224	tok_name == T_adapter_mapping			
225	tok_name == T_pin_name_mapping			
226	{			
227	{ /* handle left recursion */			
228	kids == Parse_Statement () ;			
229	return kids ;			
230	/* Parse_Statement_list */			
231				
232	static unsigned short			
233	Parse_Statement ()			
234	{ unsigned short kids = 0 ;			
235	if (tok_name == T_pin_name_mapping)			
236	{ scan () ;			
237	kids == Parse_Pin_name_mapping_list () ;			
238	bldnostrm (N_PIN_MAP , kids) ; kids = 1 ;			
239				
240				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE 5/23/89
TIME 4:42:21 pm

PAGE #
3/25

```

LINE #      SOURCE TEXT
241  else if ( tok_name == T_adapter_mapping )
242  {
243      kids == Parse_Adapter_mapping_list ( ) ;
244      bidnostrm ( M_ADAPTER , kids ) ; kids = 1 ;
245  }
246  else if ( tok_name == T_package_mapping )
247  {
248      kids == Parse_Package_mapping_list ( ) ;
249      bidnostrm ( M_PACKAGE , kids ) ; kids = 1 ;
250  }
251  else if ( tok_name == T_default_delay )
252  {
253      kids == Parse_Timing ( ) ;
254      bidnostrm ( M_DEFAULT_DELAY , kids ) ; kids = 1 ;
255  }
256  else if ( tok_name == T_delay )
257  {
258      kids == Parse_From_list ( ) ;
259      bidnostrm ( M_DELAY , kids ) ; kids = 1 ;
260  }
261  else if ( tok_name == T_ground_pin )
262  {
263      kids == Parse_Pin_corr_list ( ) ;
264      bidnostrm ( M_GROUND_PINS , kids ) ; kids = 1 ;
265  }
266  else if ( tok_name == T_power_pin )
267  {
268      kids == Parse_Pin_corr_list ( ) ;
269      bidnostrm ( M_POWER_PINS , kids ) ; kids = 1 ;
270  }
271  else if ( tok_name == T_out_pin )
272  {
273      kids == Parse_Pin_corr_list ( ) ;
274      bidnostrm ( M_OUT_PINS , kids ) ; kids = 1 ;
275  }
276  else if ( tok_name == T_mc_pin )
277  {
278      kids == Parse_Pin_corr_list ( ) ;
279      bidnostrm ( M_MC_PINS , kids ) ; kids = 1 ;
280  }
281  else if ( tok_name == T_io_pin )
282  {
283      kids == Parse_Pin_corr_list ( ) ;
284      bidnostrm ( M_IO_PINS , kids ) ; kids = 1 ;
285  }
286  else if ( tok_name == T_in_pin )
287  {
288      kids == Parse_Pin_corr_list ( ) ;
289      bidnostrm ( M_IN_PINS , kids ) ; kids = 1 ;
290  }
291  else if ( tok_name == T_initialize )
292  {
293      kids == Parse_Init_seq_list ( ) ;
294      bidnostrm ( M_INITIALIZE , kids ) ; kids = 1 ;
295  }
296  else if ( tok_name == T_technology )
297  {
298      kids == Parse_Unit ( ) ;
299      if ( tok_name == P_ID )
300      {
301          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
302      } else syntax_error ( 2 ) ;
303      kids == Parse_Opt_tech_spec_list ( ) ;
304      bidnostrm ( M_TECHNOLOGY , kids ) ; kids = 1 ;
305  }
306  else if ( tok_name == T_device_speed )
307  {
308      kids == Parse_Unit ( ) ;
309      kids == Parse_Opt_dash_unit ( ) ;
310      bidnostrm ( M_SPEED , kids ) ; kids = 1 ;
311  }
312  else if ( tok_name == T_device_sample_time )
313  {
314      kids == Parse_Unit ( ) ;
315      if ( tok_name == P_NUM )
316      {
317          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
318      } else syntax_error ( 3 ) ;
319      kids == Parse_Opt_dash_unit ( ) ;
320      bidnostrm ( M_SAMPLE_TIME , kids ) ; kids = 1 ;
321  }
322  else if ( tok_name == T_device_setup_time )
323  {
324      kids == Parse_Unit ( ) ;
325      if ( tok_name == P_NUM )
326      {
327          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
328      } else syntax_error ( 4 ) ;
329      kids == Parse_Opt_dash_unit ( ) ;
330      bidnostrm ( M_SETUP_TIME , kids ) ; kids = 1 ;
331  }
332  else if ( tok_name == T_device_hold_time )
333  {
334      kids == Parse_Unit ( ) ;
335      if ( tok_name == P_NUM )
336      {
337          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
338      } else syntax_error ( 5 ) ;
339      kids == Parse_Opt_dash_unit ( ) ;
340      bidnostrm ( M_HOLD_TIME , kids ) ; kids = 1 ;
341  }
342  else if ( tok_name == T_inhibit_timing_measurement )
343  {
344      kids == Parse_Unit ( ) ;
345      bidnostrm ( M_INH_TIM_MEASURE , kids ) ; kids = 1 ;
346  }
347  else if ( tok_name == T_disable_timing_checking )
348  {
349      kids == Parse_Unit ( ) ;
350      bidnostrm ( M_DIS_TIM_CHECK , kids ) ; kids = 1 ;
351  }
352  else if ( tok_name == T_clock_type )
353  {
354      kids == Parse_Unit ( ) ;
355      if ( tok_name == P_ID )
356      {
357          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
358      } else syntax_error ( 6 ) ;
359      kids == Parse_Opt_dash_unit ( ) ;
360      bidnostrm ( M_CLOCK_TYPE , kids ) ; kids = 1 ;
361  }
362  else if ( tok_name == T_modeler_name )
363  {
364      kids == Parse_Name ( ) ;
365      bidnostrm ( M_MODELER , kids ) ; kids = 1 ;
366  }
367  else if ( tok_name == T_report )
368  {
369      kids == Parse_Unit ( ) ;
370      if ( tok_name == P_ID )
371      {
372          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
373      } else syntax_error ( 7 ) ;
374      kids == Parse_Opt_dash_unit ( ) ;
375      bidnostrm ( M_REPORT , kids ) ; kids = 1 ;
376  }
377  else if ( tok_name == T_device_usage )
378  {
379      kids == Parse_Unit ( ) ;
380      if ( tok_name == P_ID )
381      {
382          bidtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
383      } else syntax_error ( 8 ) ;
384      kids == Parse_Opt_dash_unit ( ) ;
385      bidnostrm ( M_DEVICE_USAGE , kids ) ; kids = 1 ;
386  }

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE 5/23/89
TIME 4:42:21 pm

PAGE #
4/26

```

LINE #          SOURCE TEXT
361         if ( tok_name == P_ID )
362         { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
363           } else syntax_error ( 8 ) ;
364         bldsostrm ( M_USAGE , kids ) , kids = 1 ,
365         }
366     else if ( tok_name == T_ultra_fast )
367     { scan ( ) ;
368       bldsostrm ( M_FAST , kids ) , kids = 1 ,
369       }
370     else if ( tok_name == T_device_name )
371     { scan ( ) ;
372       kids -- Parse_Name ( ) ;
373       bldsostrm ( M_NAME , kids ) , kids = 1 ,
374       }
375     else syntax_error ( 9 ) ;
376     return kids ;
377 } /* Parse_Statement */
378
379 static unsigned short
380 Parse_Name ( )
381 { unsigned short kids = 0 ;
382   if ( tok_name == P_STR_ )
383   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
384     }
385   else if ( tok_name == P_NUM_ )
386   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
387     }
388   else if ( tok_name == P_ID )
389   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
390     }
391   else syntax_error ( 10 ) ;
392   return kids ;
393 } /* Parse_Name */
394
395 static unsigned short
396 Parse_Opt_id ( )
397 { unsigned short kids = 0 ;
398   if ( tok_name == P_ID_ )
399   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
400     }
401   return kids ;
402 } /* Parse_Opt_id */
403
404 static unsigned short
405 Parse_Unit ( )
406 { unsigned short kids = 0 ;
407   if ( tok_name == P_NUM_ )
408   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
409     if ( tok_name == P_ID_ )
410     { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
411       bldsostrm ( M_UNIT , kids ) , kids = 1 ;
412     }
413   else
414   { bldsostrm ( M_UNIT , kids ) , kids = 1 ;
415     }
416   }
417   else if ( tok_name == P_ID )
418   { bldtrm ( tok_name , tok_text ) , ++kids , scan ( ) ;
419     bldsostrm ( M_UNIT , kids ) , kids = 1 ;
420   }
421   else syntax_error ( 11 ) ;
422   return kids ;
423 } /* Parse_Unit */
424
425 static unsigned short
426 Parse_Opt_dash_unit ( )
427 { unsigned short kids = 0 ;
428   if ( tok_name == T_da )
429   { scan ( ) ;
430     kids -- Parse_Unit ( ) ;
431   }
432   return kids ;
433 } /* Parse_Opt_dash_unit */
434
435 static unsigned short
436 Parse_Opt_tech_spec_list ( )
437 { unsigned short kids = 0 ;
438   if ( tok_name == P_ID_ )
439   { kids -- Parse_Tech_spec_list ( ) ;
440     }
441   return kids ;
442 } /* Parse_Opt_tech_spec_list */
443
444 static unsigned short
445 Parse_Init_seq_list ( )
446 { unsigned short kids = 0 ;
447   kids -- Parse_Init_seq ( ) ;
448   while
449   { tok_name == P_ID ||
450     tok_name == P_STR ||
451     tok_name == P_NUM
452     }
453   { /* handle left recursion */
454     kids -- Parse_Init_seq ( ) ;
455   }
456   return kids ;
457 } /* Parse_Init_seq_list */
458
459 static unsigned short
460 Parse_Pin_corr_list ( )
461 { unsigned short kids = 0 ;
462   kids -- Parse_Pin_corr ( ) ;
463   while
464   { tok_name == P_ID ||
465     tok_name == P_STR ||
466     tok_name == P_NUM
467     }
468   { /* handle left recursion */
469     kids -- Parse_Pin_corr ( ) ;
470   }
471   return kids ;
472 } /* Parse_Pin_corr_list */
473
474 static unsigned short
475 Parse_From_list ( )
476 { unsigned short kids = 0 ;
477   kids -- Parse_From ( ) ;
478   while
479   {

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/Hparse.c	DATE 5/23/89	PAGE # 5/27
LINE #	SOURCE TEXT			
481	{ tok_name == T_from			
482	{ /* handle left recursion */			
483	kids ++ Parse_from () ;			
484	{			
485	return kids ;			
486	} /* Parse_from_list */			
487	static unsigned short			
488	Parse_Timing ()			
489	{ unsigned short kids = 0 ;			
490	{			
491	{ tok_name == P_NUM			
492	tok_name == T_ip			
493	tok_name == T_da			
494	tok_name == P_ID_			
495	{			
496	kids ++ Parse_Spec () ;			
497	if (tok_name == T_cm)			
498	{ scan () ;			
499	{			
500	{ tok_name == P_NUM			
501	tok_name == T_ip			
502	tok_name == T_da			
503	tok_name == P_ID_			
504	{			
505	kids ++ Parse_Spec () ;			
506	if (tok_name == T_cm)			
507	{ scan () ;			
508	kids ++ Parse_Spec () ;			
509	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
510	{			
511	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
512	{			
513	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
514	{			
515	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
516	{			
517	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
518	{			
519	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
520	{			
521	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
522	{			
523	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
524	{			
525	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
526	{			
527	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
528	{			
529	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
530	{			
531	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
532	{			
533	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
534	{			
535	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
536	{			
537	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
538	{			
539	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
540	{			
541	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
542	{			
543	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
544	{			
545	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
546	{			
547	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
548	{			
549	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
550	{			
551	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
552	{			
553	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
554	{			
555	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
556	{			
557	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
558	{			
559	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
560	{			
561	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
562	{			
563	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
564	{			
565	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
566	{			
567	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
568	{			
569	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
570	{			
571	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
572	{			
573	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
574	{			
575	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
576	{			
577	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
578	{			
579	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
580	{			
581	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
582	{			
583	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
584	{			
585	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
586	{			
587	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
588	{			
589	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
590	{			
591	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
592	{			
593	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
594	{			
595	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
596	{			
597	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
598	{			
599	bldnsstrm (N_TIMING , kids) , kids = 1 ;			
600	{			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE 5/23/89

PAGE #

TIME 4:42:21 pm

6/28

```

LINE # SOURCE TEXT
601 if ( tok_name == T_ip )
602 { scan ( ) ;
603 kids += Parse_Signed_num ( ) ;
604 if ( tok_name == T_rp ) scan ( ) ; else syntax_error ( 17 ) ;
605 }
606 else if ( tok_name == T_da )
607 { scan ( ) ;
608 kids += Parse_Signed_num ( ) ;
609 bldstrm ( M_NEGATIVE , kids ) ; kids = 1 ;
610 }
611 else if ( tok_name == P_NUM_ )
612 { bldtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
613 }
614 else syntax_error ( 18 ) ;
615 return kids ;
616 } /* Parse_Signed_num */
617
618 static unsigned short
619 Parse_Init_seq ( )
620 { unsigned short kids = 0 ;
621 kids += Parse_Pin_name_list ( ) ;
622 if ( tok_name == T_eq ) scan ( ) ; else syntax_error ( 19 ) ;
623 kids += Parse_Init_list ( ) ;
624 bldstrm ( M_INIT_SEQ , kids ) ; kids = 1 ;
625 return kids ;
626 } /* Parse_Init_seq */
627
628 static unsigned short
629 Parse_Pin_name_list ( )
630 { unsigned short kids = 0 ;
631 kids += Parse_Pin_name ( ) ;
632 while
633 { tok_name == T_cm
634 {
635 /* handle left recursion */
636 if ( tok_name == T_cm ) scan ( ) ; else syntax_error ( 20 ) ;
637 kids += Parse_Pin_name ( ) ;
638 }
639 return kids ;
640 } /* Parse_Pin_name_list */
641
642 static unsigned short
643 Parse_Init_list ( )
644 { unsigned short kids = 0 ;
645 kids += Parse_Init_value_list ( ) ;
646 bldstrm ( M_LIST , kids ) ; kids = 1 ;
647 return kids ;
648 } /* Parse_Init_list */
649
650 static unsigned short
651 Parse_Init_value_list ( )
652 { unsigned short kids = 0 ;
653 kids += Parse_Init_value ( ) ;
654 while
655 { tok_name == T_cm
656 {
657 /* handle left recursion */
658 if ( tok_name == T_cm ) scan ( ) ; else syntax_error ( 21 ) ;
659 kids += Parse_Init_value ( ) ;
660 }
661 return kids ;
662 } /* Parse_Init_value_list */
663
664 static unsigned short
665 Parse_Init_value ( )
666 { unsigned short kids = 0 ;
667 if ( tok_name == P_NUM_ )
668 { bldtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
669 if ( tok_name == T_ip )
670 { scan ( ) ;
671 kids += Parse_Init_list ( ) ;
672 if ( tok_name == T_rp ) scan ( ) ; else syntax_error ( 22 ) ;
673 bldstrm ( M_REPEAT , kids ) ; kids = 1 ;
674 }
675 }
676 else if ( tok_name == P_ID_ )
677 { bldtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;
678 if ( tok_name == T_ip ) scan ( ) ; else syntax_error ( 23 ) ;
679 kids += Parse_Init_list ( ) ;
680 if ( tok_name == T_rp ) scan ( ) ; else syntax_error ( 24 ) ;
681 bldstrm ( M_FEEDBACK , kids ) ; kids = 1 ;
682 }
683 else syntax_error ( 25 ) ;
684 return kids ;
685 } /* Parse_Init_value */
686
687 static unsigned short
688 Parse_Pin_corr ( )
689 { unsigned short kids = 0 ;
690 kids += Parse_Pin_name_list ( ) ;
691 if ( tok_name == T_eq ) scan ( ) ; else syntax_error ( 26 ) ;
692 kids += Parse_Names ( ) ;
693 kids += Parse_Opt_pin_attr_list ( ) ;
694 bldstrm ( M_PIN_CORR , kids ) ; kids = 1 ;
695 return kids ;
696 } /* Parse_Pin_corr */
697
698 static unsigned short
699 Parse_Names ( )
700 { unsigned short kids = 0 ;
701 kids += Parse_Name_list ( ) ;
702 bldstrm ( M_NAMES , kids ) ; kids = 1 ;
703 return kids ;
704 } /* Parse_Names */
705
706 static unsigned short
707 Parse_Opt_pin_attr_list ( )
708 { unsigned short kids = 0 ;
709 if ( tok_name == T_co )
710 { scan ( ) ;
711 kids += Parse_Pin_attr_list ( ) ;
712 }
713 return kids ;
714 } /* Parse_Opt_pin_attr_list */
715
716 static unsigned short
717 Parse_Pin_name ( )
718 { unsigned short kids = 0 ;
719 if ( tok_name == P_STP_ )
720 { bldtrm ( tok_name , tok_text ) ; ++kids ; scan ( ) ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE 5/23/89
TIME 4:42:21 pm

PAGE #
7/29

LINE # SOURCE TEXT

```

721 if
722 { tok_name == T_ls
723 {
724 kids ++ Parse_Subscript ( );
725 bldnostrm ( M_NAME, kids ); kids = 1 ;
726 }
727 }
728 else if ( tok_name == P_NUM )
729 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
730 if
731 { tok_name == T_ls
732 {
733 kids ++ Parse_Subscript ( );
734 bldnostrm ( M_PIN_NAME, kids ); kids = 1 ;
735 }
736 }
737 else if ( tok_name == P_ID )
738 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
739 if
740 { tok_name == T_ls
741 {
742 kids ++ Parse_Subscript ( );
743 bldnostrm ( M_PIN_NAME, kids ); kids = 1 ;
744 }
745 }
746 else syntax_error ( 27 );
747 return kids ;
748 } /* Parse_Pin_name */
749
750 static unsigned short
751 Parse_Subscript ( )
752 { unsigned short kids = 0 ;
753 if ( tok_name == T_ls ) scan ( ); else syntax_error ( 28 );
754 if ( tok_name == P_NUM )
755 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
756 else syntax_error ( 29 );
757 kids ++ Parse_Opt_colon_num ( );
758 if ( tok_name == T_rs ) scan ( ); else syntax_error ( 30 );
759 return kids ;
760 } /* Parse_Subscript */
761
762 static unsigned short
763 Parse_Opt_colon_num ( )
764 { unsigned short kids = 0 ;
765 if ( tok_name == T_col )
766 { scan ( );
767 if ( tok_name == P_NUM )
768 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
769 } else syntax_error ( 31 );
770 }
771 return kids ;
772 } /* Parse_Opt_colon_num */
773
774 static unsigned short
775 Parse_Name_list ( )
776 { unsigned short kids = 0 ;
777 kids ++ Parse_Name ( );
778 while
779 { tok_name == T_cm
780 }
781 { /* handle left recursion */
782 if ( tok_name == T_cm ) scan ( ); else syntax_error ( 32 );
783 kids ++ Parse_Name ( );
784 }
785 return kids ;
786 } /* Parse_Name_list */
787
788 static unsigned short
789 Parse_Pin_attr_list ( )
790 { unsigned short kids = 0 ;
791 kids ++ Parse_Pin_attr ( );
792 while
793 { tok_name == T_cm
794 }
795 { /* handle left recursion */
796 if ( tok_name == T_cm ) scan ( ); else syntax_error ( 33 );
797 kids ++ Parse_Pin_attr ( );
798 }
799 return kids ;
800 } /* Parse_Pin_attr_list */
801
802 static unsigned short
803 Parse_Pin_attr ( )
804 { unsigned short kids = 0 ;
805 if ( tok_name == P_ID )
806 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
807 if ( tok_name == T_eq )
808 { scan ( );
809 if
810 { tok_name == P_NUM ||
811 tok_name == T_lp ||
812 tok_name == T_da
813 }
814 { kids ++ Parse_Signed_num ( );
815 bldnostrm ( M_ATTR, kids ); kids = 1 ;
816 }
817 else if ( tok_name == P_ID )
818 { bldtrm ( tok_name, tok_text ); ++kids ; scan ( );
819 bldnostrm ( M_ATTR, kids ); kids = 1 ;
820 }
821 else syntax_error ( 34 );
822 }
823 }
824 else syntax_error ( 35 );
825 return kids ;
826 } /* Parse_Pin_attr */
827
828 static unsigned short
829 Parse_From ( )
830 { unsigned short kids = 0 ;
831 if ( tok_name == T_from ) scan ( ); else syntax_error ( 36 );
832 kids ++ Parse_Pin_state ( );
833 kids ++ Parse_To_list ( );
834 bldnostrm ( M_FROM, kids ); kids = 1 ;
835 return kids ;
836 } /* Parse_From */
837
838 static unsigned short
839 Parse_Pin_state ( )
840 { unsigned short kids = 0 ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hparse.c

DATE	5/23/89	PAGE #
TIME	4:42:21 pm	8/30

```

LINE #      SOURCE TEXT
841      if ( tok_name == P_ID )
842      { bldtrn ( tok_name , tok_text ) , ++kids , scan ( ) ,
843      } else syntax_error ( 37 ) ;
844      if ( tok_name == T_lp ) scan ( ) , else syntax_error ( 38 ) ,
845      kids ++ Parse_Pin_name_list ( ) ,
846      if ( tok_name == T_rp ) scan ( ) , else syntax_error ( 39 ) ,
847      bldconstrn ( M_PIN_STATE , kids ) , kids = 1 ,
848      return kids ,
849      } /* Parse_Pin_state */
850
851      static unsigned short
852      Parse_To_list ( )
853      { unsigned short kids = 0 ,
854      kids ++ Parse_To ( ) ,
855      while
856      { tok_name == T_to
857      }
858      /* handle left recursion */
859      kids ++ Parse_To ( ) ,
860      }
861      return kids ,
862      } /* Parse_To_list */
863
864      static unsigned short
865      Parse_To ( )
866      { unsigned short kids = 0 ,
867      if ( tok_name == T_to ) scan ( ) , else syntax_error ( 40 ) ,
868      kids ++ Parse_Pin_state ( ) ,
869      if ( tok_name == T_eq ) scan ( ) , else syntax_error ( 41 ) ,
870      kids ++ Parse_Timing ( ) ,
871      bldconstrn ( M_TO , kids ) , kids = 1 ,
872      return kids ,
873      } /* Parse_To */
874
875      static unsigned short
876      Parse_Spec ( )
877      { unsigned short kids = 0 ,
878      if
879      { tok_name == P_MW ||
880      tok_name == T_lp ||
881      tok_name == T_ds
882      }
883      { kids ++ Parse_Signed_num ( ) ,
884      bldconstrn ( M_SPEC , kids ) , kids = 1 ,
885      }
886      else if ( tok_name == P_ID )
887      { bldtrn ( tok_name , tok_text ) , ++kids , scan ( ) ,
888      kids ++ Parse_Signed_num ( ) ,
889      bldconstrn ( M_SPEC , kids ) , kids = 1 ,
890      }
891      else syntax_error ( 42 ) ,
892      return kids ,
893      } /* Parse_Spec */
894
895      static unsigned short
896      Parse_Package_mapping ( )
897      { unsigned short kids = 0 ,
898      kids ++ Parse_Names ( ) ,
899      if ( tok_name == T_eq ) scan ( ) , else syntax_error ( 43 ) ,
900      kids ++ Parse_Names ( ) ,
901      bldconstrn ( M_MAPPING , kids ) , kids = 1 ,
902      return kids ,
903      } /* Parse_Package_mapping */
904
905      static unsigned short
906      Parse_Adapter_mapping ( )
907      { unsigned short kids = 0 ,
908      kids ++ Parse_Names ( ) ,
909      if ( tok_name == T_eq ) scan ( ) , else syntax_error ( 44 ) ,
910      kids ++ Parse_Names ( ) ,
911      kids ++ Parse_Opt_pin_attr_list ( ) ,
912      bldconstrn ( M_MAPPING , kids ) , kids = 1 ,
913      return kids ,
914      } /* Parse_Adapter_mapping */
915
916      static unsigned short
917      Parse_Pin_name_mapping ( )
918      { unsigned short kids = 0 ,
919      kids ++ Parse_Pin_names ( ) ,
920      if ( tok_name == T_eq ) scan ( ) , else syntax_error ( 45 ) ,
921      kids ++ Parse_Pin_names ( ) ,
922      bldconstrn ( M_MAPPING , kids ) , kids = 1 ,
923      return kids ,
924      } /* Parse_Pin_name_mapping */
925
926      static unsigned short
927      Parse_Pin_names ( )
928      { unsigned short kids = 0 ,
929      kids ++ Parse_Pin_name_list ( ) ,
930      bldconstrn ( M_NAMES , kids ) , kids = 1 ,
931      return kids ,
932      } /* Parse_Pin_names */
933
934      /* end of recursive descent parser */
935

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hscan.c

DATE 5/23/89

PAGE #

TIME 4:42:23 pm

1/31

```

LINE # SOURCE TEXT
1  /* SCCS ID: Hscan.c rev 3.1, 4/24/89 at 08:00:48 */
2  /*
3   * Scanner for HSQL
4   */
5
6  #include "HSQL.h"
7  #include "common.h"
8  #include "strings.h"
9  #include "strarg.h"
10 #include "spars.h"
11
12 #define MAXTOKEN 1024
13 #define SHOWTOKEN 32
14
15 extern void lm_print_message ( ) ;
16
17 symbol
18 tok_name , /* current token name */
19 tok_text , /* current token text */
20 source , /* symbols for source file name */
21 static short nextc , /* the scanners lookahead character */
22 static char
23 *source_text , /* the source text buffer */
24 num_buf [ 12 ] , /* to print maximum token length */
25 tokens [ MAXTOKEN ] , /* the current token text buffer */
26 *cursor , /* cursor for token text */
27 static unsigned short
28 newlinesc , /* to delay line number incrementing */
29 ccount , /* comment nesting count */
30
31 static void
32 getnext ( )
33 /* gets the next character for the scanner */
34 {
35     nextc = *source_text++ ;
36     if ( nextc == '\n' )
37         ++newlinesc ;
38 } /* getnext */
39
40 void
41 init_scanner ( text )
42 char *text ;
43 /* initializes the scanner */
44 {
45     source_text = text ;
46     newlinesc = 1 ;
47     getnext ( ) ;
48 } /* init_scanner */
49
50 static void
51 lexical_error ( )
52 /* reports a lexical error, should never happen because of prescan() */
53 {
54     lm_error
55     ( 1 ,
56     /*LMSG*/"internal error: unexpected lexical error"
57     ) ;
58 } /* lexical_error */
59
60 static void
61 trace_token ( )
62 /* traces the current token (iff t_flag is on) */
63 {
64     if ( ! t_flag_on )
65         return ;
66     lm_print_message
67     ( "tok_name,tok_text=%s,%s\n" ,
68     systratr ( tok_name ) ,
69     systratr ( tok_text )
70     ) ;
71 } /* trace_token */
72
73 void
74 scan ( )
75 /* the main scanner */
76 {
77     while ( ! lm_next_line ( newlinesc ) , nextc != '\0' )
78     {
79         if ( nextc < ' ' )
80             getnext ( ) ;
81         else if
82         ( nextc >= 'A' && nextc <= 'Z' ||
83           nextc >= 'a' && nextc <= 'z' ||
84           nextc == '_' )
85         {
86             cursor = tokens ;
87             *cursor++ = nextc ;
88             while
89             ( ( getnext ( ) , nextc ) >= 'A' && nextc <= 'Z' ||
90               nextc >= 'a' && nextc <= 'z' ||
91               nextc >= '0' && nextc <= '9' ||
92               nextc == '_' )
93             {
94                 *cursor++ = nextc ;
95                 if ( cursor - tokens == MAXTOKEN - 1 )
96                     goto toolong ;
97             }
98             *cursor = '\0' ;
99             tok_text = strtostr ( tokens ) ;
100             if ( tok_text >= Q_firstreserved && tok_text <= Q_lastreserved )
101             {
102                 tok_name = tok_text ;
103                 tok_text = NULLSYM ;
104             }
105             else
106                 tok_name = P_ID_ ;
107             trace_token ( ) ;
108             return ;
109         }
110         else if ( nextc == '\\' )
111         {
112             cursor = tokens ;
113             do
114             {
115                 *cursor++ = '\\' ;
116                 if ( cursor - tokens == MAXTOKEN - 1 )
117                     goto toolong ;
118                 while ( getnext ( ) , nextc ) != '\0' && nextc != '\\'
119                 {
120                     *cursor++ = nextc ;
121                     if ( cursor - tokens == MAXTOKEN - 1 )
122                         goto toolong ;
123                 }
124                 if ( nextc == '\0' )
125                     lm_error
126                     ( 1 ,
127                     /*LMSG*/"internal error: unterminated string"
128                     ) ;
129             }
130         }
131     }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hscan.c

DATE 5/23/89

PAGE #

TIME 4:42:23 pm

2/32

```

LINE # SOURCE TEXT
121 else
122 {
123     *cursor++ = '\\';
124     if ( cursor - token == MAXTOKEN - 1 )
125         goto toolong;
126     getnext ( );
127 }
128 while ( nextc == '\\' );
129 *cursor = '\\0';
130 tok_name = P_STR;
131 tok_text = strtosym ( token );
132 trace_token();
133 return;
134 }
135 else if ( nextc == ' ' )
136 {
137     cursor = token;
138     do
139     {
140         *cursor++ = ' ';
141         if ( cursor - token == MAXTOKEN - 1 )
142             goto toolong;
143         while ( ( getnext ( ), nextc ) != '\\0' && nextc != ' ' )
144             *cursor++ = nextc;
145         if ( cursor - token == MAXTOKEN - 1 )
146             goto toolong;
147     }
148     if ( nextc == '\\0' )
149         lm_error
150             ( 1,
151               /*MSG*/"internal error: unterminated string"
152             );
153     else
154     {
155         *cursor++ = ' ';
156         if ( cursor - token == MAXTOKEN - 1 )
157             goto toolong;
158         getnext ( );
159     }
160 while ( nextc == ' ' );
161 *cursor = '\\0';
162 tok_name = P_STR;
163 tok_text = strtosym ( token );
164 trace_token();
165 return;
166 }
167 else if ( nextc >= '0' && nextc <= '9' || nextc == '.' )
168 {
169     cursor = token;
170     while
171     {
172         ( ( getnext ( ), nextc ) >= 'A' && nextc <= 'Z' ||
173           nextc >= 'a' && nextc <= 'z' ||
174           nextc >= '0' && nextc <= '9' ||
175           nextc == '.' )
176     }
177     *cursor++ = nextc;
178     if ( cursor - token == MAXTOKEN - 1 )
179         goto toolong;
180 }
181 *cursor = '\\0';
182 tok_text = strtosym ( token );
183 tok_name = P_NUM;
184 trace_token();
185 return;
186 }
187 else if ( nextc == '$' )
188 {
189     do
190     {
191         getnext ( );
192         while ( nextc <= ' ' )
193             ccoust = 0;
194         while ( nextc >= '0' && nextc <= '9' )
195             ccoust += 10;
196         ccoust += nextc - '0';
197         getnext ( );
198     }
199 while ( nextc <= ' ' && nextc != '\\n' && nextc != '\\0' );
200 getnext ( );
201 cursor = token;
202 if ( nextc != '\\n' && nextc != '\\0' )
203     do
204     {
205         *cursor++ = nextc;
206         if ( cursor - token == MAXTOKEN - 1 )
207             goto toolong;
208     }
209 while ( getnext ( ), nextc != '\\0' && nextc != '\\n' );
210 *cursor = '\\0';
211 newlibase = ccoust;
212 lm_hset_input ( syntostr ( source = strtosym ( token ) ) );
213 }
214 else if ( nextc == '{' )
215 {
216     ccoust = 0;
217     while
218     {
219         ( ( getnext ( ), nextc ) != '}' || ccoust ) &&
220         nextc != '\\0'
221     }
222     if ( nextc == '}' )
223         --ccoust;
224     else if ( nextc == '{' )
225         ++ccoust;
226     if ( nextc == '}' )
227         getnext ( );
228     else
229         lm_error
230             ( 1,
231               /*MSG*/"internal error: unterminated comment"
232             );
233 }
234 else
235 {
236     switch ( nextc )
237     {
238     case ' ':
239         tok_name = T_cm;
240         break;
241     case '{':
242         tok_name = T_lp;
243         break;
244     case '}':
245         tok_name = T_rp;
246         break;
247     case ':':
248         tok_name = T_co;
249         break;
250 }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Hscan.c

DATE	5/23/89	PAGE #
TIME	4:42:23 pm	3/33

```

LINE # SOURCE TEXT
241 case '[' :
242     tok_name = T_ls ;
243     break ;
244 case ']' :
245     tok_name = T_rs ;
246     break ;
247 case '-' :
248     tok_name = T_da ;
249     break ;
250 case '=' :
251     tok_name = T_eq ;
252     break ;
253 /* only needed in behavioral sublanguage
254 case '*' :
255     tok_name = T_pl ;
256     break ;
257 case '^' :
258     tok_name = T_up ;
259     break ;
260 case '~' :
261     tok_name = T_at ;
262     break ;
263 case '/' :
264     tok_name = T_sl ;
265     break ;
266 case 'p' :
267     tok_name = T_pc ;
268     break ;
269 case 't' :
270     tok_name = T_tf ;
271     break ;
272 case 'l' :
273     getnext ( ) ;
274     if ( nextc == 'l' )
275     { tok_name = T_lm ;
276       break ;
277     }
278     tok_name = T_lm ;
279     trace_token ( ) ;
280     return ;
281 case 'v' :
282     getnext ( ) ;
283     if ( nextc == 'v' )
284     { tok_name = T_vivl ;
285       break ;
286     }
287     tok_name = T_vl ;
288     trace_token ( ) ;
289     return ;
290 case 's' :
291     getnext ( ) ;
292     if ( nextc == 's' )
293     { tok_name = T_sseq ;
294       break ;
295     }
296     tok_name = T_sx ;
297     trace_token ( ) ;
298     return ;
299 case 'e' :
300     getnext ( ) ;
301     if ( nextc == 'e' )
302     { tok_name = T_ejseq ;
303       break ;
304     }
305     tok_name = T_ej ;
306     trace_token ( ) ;
307     return ;
308 case '<' :
309     getnext ( ) ;
310     if ( nextc == '<' )
311     { tok_name = T_lseq ;
312       break ;
313     }
314     else if ( nextc == '<' )
315     { tok_name = T_ltit ;
316       break ;
317     }
318     tok_name = T_lt ;
319     trace_token ( ) ;
320     return ;
321 case '>' :
322     getnext ( ) ;
323     if ( nextc == '>' )
324     { tok_name = T_gseq ;
325       break ;
326     }
327     else if ( nextc == '>' )
328     { tok_name = T_gtgt ;
329       break ;
330     }
331     tok_name = T_gt ;
332     trace_token ( ) ;
333     return ;
334 /* only needed in behavioral sublanguage */
335 default :
336     lexical_error ( ) ;
337     getnext ( ) ;
338     continue ;
339 }
340 getnext ( ) ;
341 trace_token ( ) ;
342 return ;
343 }
344 tok_name = END_OF_FILE ;
345 trace_token ( ) ;
346 return ;
347 toolong :
348     ( void ) sprintf ( sum_buf , "%u" , MAXTOKEN ) ;
349     token [ SHOWTOKEN ] = "\0" ;
350     lm_error
351     {
352         /*LMSG*/ token too long: %s... (must be less than %s characters)
353         , token
354         , sum_buf
355     } ;
356     tok_name = END_OF_FILE ;
357 } /* scan */
358
359 /* end of scanner for HSQL */
360

```


Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Pr_mdln.c

DATE 5/23/89

PAGE #

TIME 4:42:23 pm

1/34

LINE # SOURCE TEXT

```

1  /* SOCS_ID: Pr_mdln.c rev 3.2, 5/9/89 at 15:24:00 */
2
3  /*
4   * Processor for ENGL, MODLER SIDE
5   */
6
7  #ifdef WRITE_DEVICE
8  #include <stdio.h>
9  #endif WRITE_DEVICE
10
11 #include <setjmp.h>
12 #include <common.h>
13 #include "ENGL.h"
14 #include <strings.h>
15 #include <string.h>
16 #include "Parse.h"
17 #include "trees.h"
18 #include "Ecost.h"
19 #include "device.h"
20
21 extern void
22   init_scanner ( ) , /* scanner initializer */
23   init_constraint ( ) , /* constraint initializer */
24   print ( ) , /* parser initializer */
25   init_the ( ) ,
26   init_the_device ( ) ,
27   free_the ( ) ,
28   free_the_device ( ) ,
29   extern struct dev *constrain ( ) , /* the constraint */
30   extern struct device *gen ( ) , /* the device generator */
31   extern tree parse ( ) , /* the parser */
32 #ifdef WRITE_DEVICE
33   extern void print_dev ( ) , /* the device printer */
34   extern void print_map ( ) , /* the device printer */
35   extern void dump_power_ground ( ) , /* power/ground pin dumper */
36   extern ushort write_device ( ) , /* the device writer */
37 #endif WRITE_DEVICE
38
39 jmp_buf backend_jump ;
40
41 void
42 syntax_error ( s )
43   unsigned short s ;
44 /* reports a syntax error via error() */
45 {
46   char *mess ;
47   switch ( s )
48   {
49     case 1 :
50       mess =
51         /*MSG*/"end of file or statement reserved word expected"
52       ;
53       break ;
54     case 2 :
55       mess =
56         /*MSG*/"identifier expected after 'technology'"
57       ;
58       break ;
59     case 3 :
60       mess =
61         /*MSG*/"number expected after 'device_sample_time'"
62       ;
63       break ;
64     case 4 :
65       mess =
66         /*MSG*/"number expected after 'device_sample_time'"
67       ;
68       break ;
69     case 5 :
70       mess =
71         /*MSG*/"number expected after 'device_hold_time'"
72       ;
73       break ;
74     case 6 :
75       mess =
76         /*MSG*/"identifier expected after 'clock_type'"
77       ;
78       break ;
79     case 7 :
80       mess =
81         /*MSG*/"identifier expected after 'report'"
82       ;
83       break ;
84     case 8 :
85       mess =
86         /*MSG*/"identifier expected after 'device_usage'"
87       ;
88       break ;
89     case 10 :
90     case 27 :
91       mess =
92         /*MSG*/"identifier, number, or string expected"
93       ;
94       break ;
95     case 11 :
96     case 25 :
97       mess =
98         /*MSG*/"identifier or number expected"
99       ;
100      break ;
101     case 12 :
102       mess =
103         /*MSG*/"timing specification expected"
104       ;
105      break ;
106     case 13 :
107       mess =
108         /*MSG*/"timing specification expected after ','"
109       ;
110      break ;
111     case 14 :
112     case 20 :
113     case 21 :
114     case 28 :
115     case 32 :
116     case 33 :
117       mess =
118         /*MSG*/"internal error: impossible syntax error"
119       ;
120      break ;
121     case 15 :

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Pr_mdir.c

DATE	5/23/89	PAGE #
TIME	4:42:23 pm	2/35

```

LINE #          SOURCE TEXT
121      case 15 :
122      case 17 :
123      mess =
124      /*LMSC*/"identifier expected"
125      /
126      break ;
127      case 16 :
128      mess =
129      /*LMSC*/"'" expected after identifier"
130      /
131      break ;
132      case 17 :
133      mess =
134      /*LMSC*/"'" expected after signed number"
135      /
136      break ;
137      case 18 :
138      mess =
139      /*LMSC*/"signed number expected"
140      /
141      break ;
142      case 19 :
143      case 26 :
144      case 45 :
145      mess =
146      /*LMSC*/"'" expected after pin name list"
147      /
148      break ;
149      case 22 :
150      case 24 :
151      mess =
152      /*LMSC*/"'" expected after initialization list"
153      /
154      break ;
155      case 23 :
156      case 38 :
157      mess =
158      /*LMSC*/"(' expected after identifier"
159      /
160      break ;
161      case 29 :
162      mess =
163      /*LMSC*/"number expected after '['"
164      /
165      break ;
166      case 30 :
167      mess =
168      /*LMSC*/"'" expected after number"
169      /
170      break ;
171      case 31 :
172      mess =
173      /*LMSC*/"number expected after ':'"
174      /
175      break ;
176      case 34 :
177      mess =
178      /*LMSC*/"identifier or signed number expected after '"'"
179      /
180      break ;
181      case 36 :
182      mess =
183      /*LMSC*/"from expected"
184      /
185      break ;
186      case 39 :
187      mess =
188      /*LMSC*/"'" expected after pin name list"
189      /
190      break ;
191      case 40 :
192      mess =
193      /*LMSC*/"to expected"
194      /
195      break ;
196      case 41 :
197      mess =
198      /*LMSC*/"'" expected after pin state"
199      /
200      break ;
201      case 42 :
202      mess =
203      /*LMSC*/"identifier or signed number expected"
204      /
205      break ;
206      case 43 :
207      case 44 :
208      mess =
209      /*LMSC*/"'" expected after name list"
210      /
211      break ;
212      default :
213      mess =
214      /*LMSC*/"internal error: unknown syntax error"
215      /
216      break ;
217      }
218      lm_error
219      ( 1 ,
220      /*LMSC*/"syntax error: ts"
221      mess
222      ) ;
223      /* syntax_error */
224      /
225      static void
226      init_MMDL ( source )
227      char *source ;
228      /* initializes the backend */
229      {
230      init_the ( ) ;
231      init_the_device ( ) ;
232      pinit ( ) ;
233      init_scanner ( source ) ;
234      init_constraint ( ) ;
235      /* init_MMDL */
236      }
237      static void
238      show_stats ( T )
239      tree T ;
240      {
241      if
242      ( lm_M_flag_on ( ) )

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/Pr_mdlr.c

DATE 5/23/89 PAGE #
TIME 4:42:23 pm 3/36

```

LINE # SOURCE TEXT
241     lm_Ep_flag_on ( ) ||
242     lm_Es_flag_on ( ) ||
243     lm_Et_flag_on ( )
244     } void ) lm_print_message ( "\n" );
245     } void ) lm_print_message ( "id nodes in the AST\n", T );
246     } /* show_stats */
247
248 static void
249 free_all ( )
250 {
251     free_the ( ) ;
252     free_the_device ( ) ;
253     strfree ( ) ;
254     treefree ( ) ;
255     } /* free_all */
256
257 void
258 lm_Ffail ( )
259 {
260     _longjmp ( backend_jump , 1 );
261     } /* lm_Ffail */
262
263 struct device *
264 backend_pass ( source )
265 char *source ;
266 /* make the backend pass (scan, parse, constrain, generate, write) */
267 {
268     struct dev *dev ;
269     struct device *device ;
270     if ( _setjmp ( backend_jump ) )
271     {
272         free_all ( ) ;
273         return NULL ;
274     }
275     if ( lm_El_flag_on ( ) )
276     {
277         ( void ) lm_print_message ( "ls", source );
278         init_EWML ( source );
279         AST = parse ( ) ;
280         if ( lm_Errors ( ) )
281         {
282             strfree ( ) ;
283             treefree ( ) ;
284             return NULL ;
285         }
286     }
287     if ( lm_Ep_flag_on ( ) )
288     {
289         if ( lm_El_flag_on ( ) )
290             ( void ) lm_print_message ( "\n" );
291         printtree ( AST );
292     }
293     if ( lm_Es_flag_on ( ) )
294     {
295         if ( lm_El_flag_on ( ) || lm_Ep_flag_on ( ) )
296             ( void ) lm_print_message ( "\n" );
297         strstat ( ) ;
298     }
299     if
300     {
301         ( dev = constrain ( AST ) ) != NULL &&
302         ( lm_Eg_flag_on ( ) ? ( show_stats ( AST ), 1 ) : 1 ) &&
303         # ifdef WRITE_DEVICE
304         ( lm_Ec_flag_on ( ) ? ( dump_power_ground ( dev , AST ), 1 ) : 1 ) &&
305         ( lm_Ed_flag_on ( ) ? ( print_dev ( dev ), 1 ) : 1 ) &&
306         ( lm_Ea_flag_on ( ) ? ( print_map ( AST ), 1 ) : 1 ) &&
307         # endif WRITE_DEVICE
308         ( device = gen ( dev ) ) != NULL
309         # ifdef WRITE_DEVICE
310         &&
311         {
312             lm_Ev_flag_on ( ) ||
313             lm_Ea_flag_on ( ) ||
314             write_device ( stdout , device )
315         }
316     }
317     # endif WRITE_DEVICE
318     return device ;
319     return NULL ;
320     } /* backend_pass */
321
322 /* end of Processor for EWML, MODELER SIDE */
323

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/extract_dev.c

DATE	5/23/89	PAGE #
TIME	4:42:23 pm	1/37

```

1  LINE # SOURCE TEXT
2  /* SOCS_ID: extract_dev.c rev 3.1, 4/24/89 at 08:01:09 */
3  /* EMBL Device Definition Extractor */
4  #include "device.h"
5  #include <stdio.h>
6
7  #define BUF_SIZE 10240 /* initial buffer size */
8  #define BUF_INCR 5120 /* buffer size increment */
9
10 extern char *lm_malloc ( ), *lm_realloc ( );
11
12 static char *the_buf; /* the expanding buffer */
13 unsigned long buf_len; /* the current buffer length */
14 unsigned long buf_size; /* the current maximum buffer length */
15
16 static void
17 init_buf ( )
18 {
19     buf_len = 0;
20     the_buf = lm_malloc ( BUF_SIZE );
21     if ( the_buf == NULL )
22     { /* Report error */
23         buf_size = 0;
24     }
25     else
26     {
27         buf_size = BUF_SIZE;
28     } /* init_buf */
29
30 static void
31 extend_buf ( )
32 {
33     if ( buf_size == 0 )
34     {
35         return;
36     }
37     the_buf = lm_realloc ( the_buf, ( unsigned ) ( buf_size + BUF_INCR ) );
38     if ( the_buf == NULL )
39     { /* Report error */
40         buf_size = 0;
41     }
42     else
43     {
44         buf_size += BUF_INCR;
45     } /* extend_buf */
46
47 static void
48 print_str ( s )
49 char *s
50 {
51     unsigned short len;
52     len = strlen ( s );
53     if ( len + buf_len >= buf_size )
54     {
55         extend_buf ( );
56     }
57     if ( buf_size == 0 )
58     {
59         return;
60     }
61     ( void ) strcpy ( the_buf + buf_len, s );
62     buf_len += len;
63 } /* print_str */
64
65 static void
66 print_num ( n )
67 long n
68 {
69     unsigned short len;
70     char s [ 12 ];
71     /* ( void ) sprintf ( s, "%ld", n );
72     /*
73     (char t[12]);
74     unsigned short i;
75     long o;
76     o=n;
77     t[11]='\0';
78     i=10;
79     if(o>0)
80     t[i]='0';
81     else
82     {while(o)
83     {t[i]=o%10+'0';
84     o/=10;
85     --i;
86     }
87     ++i;
88     }
89     (void)strcpy(s,t+i);
90 }
91 len = strlen ( s );
92 if ( len + buf_len >= buf_size )
93     extend_buf ( );
94 if ( buf_size == 0 )
95     return;
96 ( void ) strcpy ( the_buf + buf_len, s );
97 buf_len += len;
98 } /* print_num */
99
100 static void
101 print_flt ( f )
102 float f
103 {
104     unsigned short len;
105     char s [ 36 ];
106     ( void ) sprintf ( s, "%g", f );
107     len = strlen ( s );
108     if ( len + buf_len >= buf_size )
109     {
110         extend_buf ( );
111     }
112     if ( buf_size == 0 )
113     {
114         return;
115     }
116     ( void ) strcpy ( the_buf + buf_len, s );
117     buf_len += len;
118 } /* print_flt */
119
120 static char *
121 dev_of ( type )
122 unsigned short type
123 {
124     switch ( type )
125     {
126     case PUBLIC:
127         return "public";
128     case PRIVATE:
129         return "private";
130     default:
131         return "bad";
132     }
133 } /* dev_of */
134
135 static char *
136 clock_of ( type )
137 unsigned short type

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/extract_dev.c

DATE	5/23/89	PAGE #
TIME	4:42:23 pm	2/38

```

121  { switch ( type )
122  { case INTERNAL :
123      return "internal" ;
124      case EXT1 :
125          return "external_clock0" ;
126      case EXT2 :
127          return "external_clock1" ;
128      default :
129          return "bad" ;
130  }
131  } /* clock_of */
132
133  static char *
134  state_of ( type )
135  unsigned short type ;
136  { switch ( type )
137  { case NONE :
138      return "none" ;
139      case LOW :
140      return "low" ;
141      case HIGH :
142      return "high" ;
143      case VALID :
144      return "valid" ;
145      case FLOAT :
146      return "float" ;
147      case ANY :
148      return "any" ;
149      default :
150      return "bad" ;
151  }
152  } /* state_of */
153
154  static char *
155  class_of ( type , format )
156  unsigned short type ;
157  unsigned short format ;
158  { switch ( type )
159  { case EVAL :
160      return "eval" ;
161      case STORE :
162      switch ( format )
163      { case DMR2 :
164          return "store" ;
165          case R1 :
166          return "edge_rise" ;
167          case R0 :
168          return "edge_fall" ;
169          default :
170          return "badformat" ;
171      }
172      default :
173      return "bad" ;
174  }
175  } /* class_of */
176
177  static char *
178  drive_of ( type )
179  unsigned short type ;
180  { switch ( type )
181  { case S_DRIVE :
182      return "hard" ;
183      case M_DRIVE :
184      return "medium" ;
185      case M_DRIVE :
186      return "hard_medium" ;
187      default :
188      return "bad" ;
189  }
190  } /* drive_of */
191
192  static unsigned short
193  soft_drive_of ( val , w1 , w2 , w3 , w4 )
194  unsigned short
195  val ;
196  w1 ,
197  w2 ,
198  w3 ,
199  w4 ;
200  { return
201  { val & 8 ? w4 : 0 } +
202  { val & 4 ? w3 : 0 } +
203  { val & 2 ? w2 : 0 } +
204  { val & 1 ? w1 : 0 } ;
205  } /* soft_drive_of */
206
207  static char *
208  pin_name ( name )
209  char *name ;
210  { static char *str , buf [ 1024 ] , c [ 2 ] ;
211    str = name ;
212    while ( *str != '\0' )
213    {
214        if ( *str >= 'A' && *str <= 'Z' ||
215            *str >= 'a' && *str <= 'z' ||
216            *str >= '0' && *str <= '9' ||
217            *str == '-' || *str == '.' )
218            ++str ;
219        else
220            break ;
221    }
222    if ( *str == '\0' )
223        return name ;
224    c [ 1 ] = '\0' ;
225    ( void ) strcpy ( buf , "" ) ;
226    for ( str = name , *str != '\0' , ++str )
227    { *c = *str ;
228      ( void ) strcat ( buf , c ) ;
229      if ( *c == '\n' )
230          ( void ) strcat ( buf , c ) ;
231    }
232    ( void ) strcat ( buf , "" ) ;
233    return buf ;
234  } /* pin_name */
235
236  static unsigned short
237  seen_pin ( pin_table , i )
238  PIN_SPEC *pin_table ;
239  unsigned short i ;
240  { char *name ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/extract_dev.c

DATE 5/23/89 PAGE #
TIME 4:42:23 pm 3/39

```

LINE # SOURCE TEXT
241 unsigned short dir ;
242 name = pin_table [ i ] . pin_name ;
243 dir = pin_table [ i ] . direction ;
244 while ( 1 == )
245 {
246     ( pin_table [ i ] . direction == dir &&
247     strcmp ( pin_table [ i ] . pin_name , name ) == 0
248     )
249     return 1 ;
250     return 0 ;
251 } /* seen_pin */
252
253 static void
254 get_more_pins ( pin_table , i , pins )
255 PIN_SPEC *pin_table ;
256 unsigned short i , pins ;
257 {
258     char *name ;
259     unsigned short dir ;
260     name = pin_table [ i ] . pin_name ;
261     dir = pin_table [ i ] . direction ;
262     for ( ++i , i < pins ; ++i )
263     {
264         ( pin_table [ i ] . direction == dir &&
265         strcmp ( pin_table [ i ] . pin_name , name ) == 0
266         )
267         {
268             print_str ( "-" ) ;
269             print_str ( pin_name ( pin_table [ i ] . pin_number ) ) ;
270         }
271     } /* get_more_pins */
272
273 static unsigned short
274 seen_delay ( pins , i , j )
275 PIN_SPEC *pins ;
276 unsigned short i , j ;
277 {
278     unsigned short ii , jj , pin , state ;
279     pin = pins [ i ] . delay_table [ j ] . minor_pin ;
280     state = pins [ i ] . delay_table [ j ] . minor_state ;
281     for ( ii = 0 ; ii < i ; ++ii )
282     {
283         if ( pins [ ii ] . direction == OUT || pins [ ii ] . direction == IO )
284         {
285             for ( jj = 0 ; jj < pins [ ii ] . delay_cnt ; ++jj )
286             {
287                 ( pin == pins [ ii ] . delay_table [ jj ] . minor_pin &&
288                 state == pins [ ii ] . delay_table [ jj ] . minor_state
289                 )
290                 return 1 ;
291             }
292             for ( jj = 0 ; jj < j ; ++jj )
293             {
294                 ( pin == pins [ i ] . delay_table [ jj ] . minor_pin &&
295                 state == pins [ i ] . delay_table [ jj ] . minor_state
296                 )
297                 return 1 ;
298             }
299             return 0 ;
300         }
301     } /* seen_delay */
302
303 static void
304 do_to_delay ( pin , delay , stype )
305 PIN_SPEC *pin ;
306 TIMING_SPEC *delay ;
307 MIN_TTP_MAX *stype ;
308 {
309     print_str ( "to " ) ;
310     print_str ( state_of ( delay -> major_state ) ) ;
311     print_str ( "(" ) ;
312     print_str ( pin_name ( pin -> pin_name ) ) ;
313     print_str ( " " ) ;
314     print_flt ( ( float ) stype [ delay -> stype_index ] . minimum / 1000 ) ;
315     print_str ( " " ) ;
316     print_flt ( ( float ) stype [ delay -> stype_index ] . typical / 1000 ) ;
317     print_str ( " " ) ;
318     print_flt ( ( float ) stype [ delay -> stype_index ] . maximum / 1000 ) ;
319     print_str ( " " ) ;
320 } /* do_to_delay */
321
322 static void
323 get_more_to_delays ( pins , i , pin_cnt , j , stype )
324 PIN_SPEC *pins ;
325 unsigned short i , pin_cnt , j ;
326 MIN_TTP_MAX *stype ;
327 {
328     unsigned short pin , state ;
329     pin = pins [ i ] . delay_table [ j ] . minor_pin ;
330     state = pins [ i ] . delay_table [ j ] . minor_state ;
331     for ( ++j , j < pins [ i ] . delay_cnt ; ++j )
332     {
333         ( pin == pins [ i ] . delay_table [ j ] . minor_pin &&
334         state == pins [ i ] . delay_table [ j ] . minor_state
335         )
336         do_to_delay ( pins + i , pins [ i ] . delay_table + j , stype ) ;
337     }
338     for ( ++i , i < pin_cnt ; ++i )
339     {
340         if ( pins [ i ] . direction == OUT || pins [ i ] . direction == IO )
341         {
342             for ( j = 0 ; j < pins [ i ] . delay_cnt ; ++j )
343             {
344                 ( pin == pins [ i ] . delay_table [ j ] . minor_pin &&
345                 state == pins [ i ] . delay_table [ j ] . minor_state
346                 )
347                 do_to_delay ( pins + i , pins [ i ] . delay_table + j , stype ) ;
348             }
349         }
350     } /* get_more_to_delays */
351
352 char *
353 extract_device ( device )
354 DEVICE_SPEC *device ;
355 {
356     unsigned short
357     i ,
358     j ,
359     comma ,
360     seen ;
361     PIN_SPEC *pin ;
362     TIMING_SPEC *delay ;
363     double device_isl = -1 ;
364     if ( device == NULL )
365         return NULL ;
366     init_buf ( ) ;
367     print_str ( "device_name " ) ;
368     print_str ( pin_name ( device -> device_name ) ) ;
369     print_str ( "device_usage " ) ;
370     print_str ( dev_of ( device -> device_type ) ) ;
371     if ( device -> modeler_name != NULL )
372     {
373         print_str ( "modeler_name " ) ;
374         print_str ( device -> modeler_name ) ;
375     }
376     print_str ( "clock_type " ) ;
377     print_str ( clock_of ( device -> clock_type ) ) ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/extract_dev.c

DATE 5/23/89
TIME 4:42:23 pm

PAGE #
4/40

LINE # SOURCE TEXT

```

361 print_str ( "device speed " );
362 if ( device -> clock_period1 >= 1000000 )
363 { print_flt ( ( float ) device -> clock_period1 / 1000000 );
364   print_str ( " ns" );
365 }
366 else
367 { print_flt ( ( float ) device -> clock_period1 / 1000 );
368   print_str ( " ns" );
369 }
370 if ( device -> clock_period2 == -1 )
371   print_str ( "infinity" );
372 else if ( device -> clock_period2 > 1000000 )
373 { print_flt ( ( float ) device -> clock_period2 / 1000000 );
374   print_str ( " ns" );
375 }
376 else
377 { print_flt ( ( float ) device -> clock_period2 / 1000 );
378   print_str ( " ns" );
379 }
380 if ( device -> ultra_fast )
381   print_str ( "ultra_fast" );
382 if ( device -> dis_tlm_check )
383   print_str ( "disable_timing_checking" );
384 if ( device -> inh_tlm_measure )
385   print_str ( "inhibit_timing_measurement" );
386 if ( device -> report_misr )
387   print_str ( "report_missing_delays on" );
388 else
389   print_str ( "report_missing_delays off" );
390 if ( device -> report_issc )
391   print_str ( "report_is_state_changes on" );
392 for ( i = 0 ; device_iol == -1 && i < device -> pin_cnt ; ++i )
393 { if ( device -> pin_table [ i ] . direction == OUT )
394   device_iol =
395     ( float ) soft_drive_of
396       ( device -> pin_table [ i ] . s_drive_low ,
397         200 ,
398         500 ,
399         1200 ,
400         3000 )
401       / 1000 ;
402   if ( device_iol == -1 )
403     device_iol = 0.5 ;
404   print_str ( "technology custom vcc" );
405   print_flt ( ( float ) device -> vcc / 1000 );
406   print_str ( " vth" );
407   print_flt ( ( float ) device -> vth / 1000 );
408   print_str ( " vthn" );
409   print_flt ( ( float ) device -> vthn / 1000 );
410   print_str ( " vthp" );
411   print_flt ( ( float ) device -> vthp / 1000 );
412   print_str ( " vtl" );
413   print_flt ( ( float ) device -> vtl / 1000 );
414   print_str ( " vthn" );
415   print_flt ( ( float ) device -> vthn / 1000 );
416   print_str ( " val" );
417   print_flt ( ( float ) device -> val / 1000 );
418   print_str ( " iil" );
419   print_flt ( device_iol / 2 );
420   print_str ( " iih" );
421   print_flt ( device_iol / 2 );
422   print_str ( " iol" );
423   print_flt ( device_iol );
424   print_str ( " ioh" );
425   print_flt ( device_iol );
426   print_str ( " n" );
427   if ( device -> five_state_sample )
428   { print_str ( "device_sample_time " );
429     print_flt ( ( float ) device -> five_state_sample / 1000 );
430     print_str ( " n" );
431   }
432   /* what about two-state sample? */
433   if ( device -> setup_time != -1 )
434   { print_str ( "device_setup_time " );
435     print_flt ( ( float ) device -> setup_time / 1000 );
436     print_str ( " n" );
437   }
438   if ( device -> hold_time != -1 )
439   { print_str ( "device_hold_time " );
440     print_flt ( ( float ) device -> hold_time / 1000 );
441     print_str ( " n" );
442   }
443   seen = 0 ;
444   for ( i = 0 ; i < device -> pin_cnt ; ++i )
445   { pin = device -> pin_table + i ;
446     if ( pin -> direction == IN )
447     { if ( ! seen )
448       { print_str ( "in_pin" );
449         seen = 1 ;
450       }
451       print_str ( pin_name ( pin -> pin_name ) );
452       print_str ( pin_name ( pin -> pin_number ) );
453       if ( comma = pin -> pin_class != DATA )
454       { print_str ( " " );
455         print_str ( class_of ( pin -> pin_class , pin -> clk_format ) );
456       }
457       if ( pin -> kept_alive )
458       { print_str ( comma ? "keepalive" : "keepalive" );
459         print_str ( " n" );
460       }
461     }
462     seen = 0 ;
463     for ( i = 0 ; i < device -> pin_cnt ; ++i )
464     { pin = device -> pin_table + i ;
465       if ( pin -> direction == OUT )
466       { if ( ! seen )
467         { print_str ( "out_pin" );
468           seen = 1 ;
469         }
470         print_str ( pin_name ( pin -> pin_name ) );
471         print_str ( " " );
472         print_str ( pin_name ( pin -> pin_number ) );
473         if ( comma = pin -> pin_class != FEEDBACK )
474         { print_str ( " " );
475           print_str ( "feedback" );
476         }
477         if ( pin -> pulled_up )
478         { print_str ( comma ? "pull_up" : "pull_up" );
479           print_str ( " n" );
480         }
481         else if ( pin -> pulled_down )

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/extract_dev.c

DATE	5/23/89	PAGE #
TIME	4:42:23 pm	5/41

```

LINE # SOURCE TEXT
481 print_str ( comma ? ",pull_down" : ":pull_down" );
482 print_str ( "\n" );
483 }
484 }
485 seen = 0;
486 for ( i = 0; i < device->pin_cnt; ++i )
487 { pin = device->pin_table + i;
488   if ( pin->direction == IO )
489   { if ( ! seen )
490     { print_str ( "io_pin\n" );
491       seen = 1;
492     }
493     print_str ( pin_name ( pin->pin_name ) );
494     print_str ( "\n" );
495     print_str ( pin_name ( pin->pin_name ) );
496     comma = pin->pin_class != DATA;
497     if ( comma )
498     { print_str ( ":", " " );
499       print_str ( "class_of ( pin->pin_class, pin->clk_format ) );
500     }
501   }
502   if ( pin->feeds_back )
503   { print_str ( comma ? ",feedback" : ":feedback" );
504     comma = 1;
505   }
506   if ( pin->kept_alive )
507   { print_str ( comma ? ",keepalive" : ":keepalive" );
508     comma = 1;
509   }
510   if ( pin->in_seq_drive !=
511     ( device->micro_fast ? EM_DRIVE : M_DRIVE ) )
512   { print_str ( comma ? ",in_sequence" : ":in_sequence" );
513     print_str ( drive_of ( pin->in_seq_drive ) );
514     comma = 1;
515   }
516   if ( pin->last_cyc_drive !=
517     ( device->micro_fast ? EM_DRIVE : M_DRIVE ) ||
518     pin->last_cyc_drive != NO_DRIVE )
519   { print_str ( comma ? ",last_cycle" : ":last_cycle" );
520     print_str ( drive_of ( pin->last_cyc_drive ) );
521     comma = 1;
522   }
523   print_str ( comma ? ",tilt" : ":tilt" );
524   print_str ( "\n" );
525   if ( float ) soft_drive_of
526   { ( float ) soft_drive_of
527     ( pin->s_drive_low,
528       100,
529       500,
530       1200,
531       3000
532     ) / 1000;
533   }
534   print_str ( "\n" );
535   if ( float ) soft_drive_of
536   { ( float ) soft_drive_of
537     ( pin->s_drive_hi,
538       100,
539       500,
540       1200,
541       3000
542     ) / 1000;
543   }
544   if ( pin->pulled_up )
545     print_str ( "pull_up" );
546   else if ( pin->pulled_down )
547     print_str ( "pull_down" );
548   print_str ( "\n" );
549 }
550 }
551 for ( i = 0; i < device->pin_cnt; ++i )
552 { pin = device->pin_table + i;
553   if ( pin->direction == POWER )
554   { if ( ! seen_pin ( device->pin_table, i ) )
555     { print_str ( "power_pin\n" );
556       print_str ( pin_name ( pin->pin_name ) );
557       print_str ( "\n" );
558       print_str ( pin_name ( pin->pin_name ) );
559       get_more_pins ( device->pin_table, i, device->pin_cnt );
560       print_str ( "\n" );
561     }
562   }
563   else if ( pin->direction == GROUND )
564   { if ( ! seen_pin ( device->pin_table, i ) )
565     { print_str ( "ground_pin\n" );
566       print_str ( pin_name ( pin->pin_name ) );
567       print_str ( "\n" );
568       print_str ( pin_name ( pin->pin_name ) );
569       get_more_pins ( device->pin_table, i, device->pin_cnt );
570       print_str ( "\n" );
571     }
572   }
573   else if ( pin->direction == NC )
574   { if ( ! seen_pin ( device->pin_table, i ) )
575     { print_str ( "nc_pin\n" );
576       print_str ( pin_name ( pin->pin_name ) );
577       print_str ( "\n" );
578       print_str ( pin_name ( pin->pin_name ) );
579       get_more_pins ( device->pin_table, i, device->pin_cnt );
580       print_str ( "\n" );
581     }
582   }
583 }
584 print_str ( "package_mapping\n" );
585 for ( i = 0; i < device->pin_cnt; ++i )
586 { pin = device->pin_table + i;
587   if ( pin->direction != NONE )
588   { print_str ( pin_name ( pin->pin_name ) );
589     print_str ( "\n" );
590     print_str ( ( long ) i );
591     print_str ( "\n" );
592   }
593 }
594 print_str ( "adapter_mapping\n" );

```


Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/extract_dev.c

DATE

5/23/89

PAGE #

TIME 4:42:23 pm

6/42

```

LINE # SOURCE TEXT
601 } = 0;
602 for ( i = 0; i < device -> pin_cnt; ++i )
603 { pin = device -> pin_table + i;
604   if ( pin -> direction != NONE )
605     { print_sum ( ( long ) i );
606       print_str ( "-" );
607       print_sum ( ( long ) i );
608       if ( pin -> trace_delay )
609         { print_str ( "trace_delay=" );
610           print_flt ( ( float ) pin -> trace_delay / 1000 );
611         }
612       print_str ( "\n" );
613       if ( pin -> pin_alias != NULL )
614         j = 1;
615     }
616   if ( j )
617     { print_str ( "pin_name_mapping\n" );
618       for ( i = 0; i < device -> pin_cnt; ++i )
619       { pin = device -> pin_table + i;
620         if ( pin -> direction != NONE && pin -> pin_alias != NULL )
621           { print_str ( pin_name ( pin -> pin_alias ) );
622             print_str ( "-" );
623             print_str ( pin_name ( pin -> pin_name ) );
624             print_str ( "\n" );
625           }
626         }
627     }
628 }
629 for ( i = 0; i < device -> seq_cnt; ++i )
630 { if ( i == 0 )
631   { print_str ( "initialize\n" );
632     print_str ( "pin_name\n" );
633     { pin_name
634       { device -> pin_table
635         { device -> seq_table [ i ] . pin_number
636           . pin_name
637         }
638       }
639     }
640     print_str ( "device -> seq_table [ i ] . seq_str );
641     print_str ( "\n" );
642   }
643   seen = 0;
644   if ( device -> use_default )
645     { print_str ( "default\n" );
646       print_flt ( ( float ) device -> default_delay . minimum / 1000 );
647       print_str ( ", " );
648       print_flt ( ( float ) device -> default_delay . typical / 1000 );
649       print_str ( ", " );
650       print_flt ( ( float ) device -> default_delay . maximum / 1000 );
651       print_str ( "\n" );
652     }
653   for ( i = 0; i < device -> pin_cnt; ++i )
654   { pin = device -> pin_table + i;
655     if ( pin -> direction == OUT || pin -> direction == IO )
656       { for ( j = 0; j < pin -> delay_cnt; ++j )
657         { delay = pin -> delay_table + j;
658           if ( ! seen_delay ( device -> pin_table , i , j ) )
659             { if ( ! seen )
660               { print_str ( "delay\n" );
661                 seen = 1;
662               }
663               print_str ( "from " );
664               print_str ( state_of ( delay -> minor_state ) );
665               print_str ( " (" );
666               print_str ( "pin_name\n" );
667               { pin_name
668                 { device ->
669                   pin_table [ delay -> minor_pin ] . pin_name
670                 }
671               }
672               print_str ( ")\n" );
673               do_to_delay ( pin , delay , device -> mtyp_table );
674               get_more_to_delays
675               { device -> pin_table ,
676                 i ,
677                 device -> pin_cnt ,
678                 device -> mtyp_table
679               }
680             }
681           }
682         }
683     }
684   }
685   return buf_size > the_buf : NULL ;
686 } /* extract_device */
687
688 /* end of BML Device Definition Extractor */
689

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/free_dev.c

DATE

5/23/89

PAGE #

TIME

4:42:24 pm

1/43

```

LINE # SOURCE TEXT
1  /* SCOS_ID: free_dev.c rev 3.1, 4/24/89 at 08:01:16 */
2  /* routine to free a NML device */
3  #include "device.h"
4  extern void lm_free ( ) ;
5
6  static void
7  free_pin_table ( device )
8  struct device *device ;
9  /* frees the pin table of the passed device */
10 {
11     unsigned short i ;
12     if ( device->pin_table == NULL )
13         return ;
14     for ( i = 0 ; i < device->pin_cnt ; ++i )
15     {
16         if ( device->pin_table [ i ] . pin_name != NULL )
17             ( void ) lm_free ( device->pin_table [ i ] . pin_name ) ;
18         if ( device->pin_table [ i ] . pin_number != NULL )
19             ( void ) lm_free ( device->pin_table [ i ] . pin_number ) ;
20         if ( device->pin_table [ i ] . pin_alias != NULL )
21             ( void ) lm_free ( device->pin_table [ i ] . pin_alias ) ;
22     }
23     ( void ) lm_free ( ( char * ) device->pin_table ) ;
24 } /* free_pin_table */
25
26 static void
27 free_seq_table ( device )
28 struct device *device ;
29 /* frees the initialization sequence of the passed device */
30 {
31     unsigned short i ;
32     char *bits ;
33     if ( device->seq_table == NULL )
34         return ;
35     for ( i = 0 ; i < device->seq_cnt ; ++i )
36     {
37         bits = device->seq_table [ i ] . pre_bits ;
38         if ( bits != NULL )
39             ( void ) lm_free ( bits ) ;
40         bits = device->seq_table [ i ] . fd_bits ;
41         if ( bits != NULL )
42             ( void ) lm_free ( bits ) ;
43         bits = device->seq_table [ i ] . post_bits ;
44         if ( bits != NULL )
45             ( void ) lm_free ( bits ) ;
46         bits = device->seq_table [ i ] . seq_str ;
47         if ( bits != NULL )
48             ( void ) lm_free ( bits ) ;
49     }
50     ( void ) lm_free ( ( char * ) device->seq_table ) ;
51 } /* free_seq_table */
52
53 static void
54 free_timing_table ( device )
55 struct device *device ;
56 /* frees the timing table of the passed device */
57 {
58     if ( device->timing_table != NULL )
59         ( void ) lm_free ( ( char * ) device->timing_table ) ;
60 } /* free_timing_table */
61
62 static void
63 free_stym_table ( device )
64 struct device *device ;
65 /* frees the min-tyr-max table of the passed device */
66 {
67     if ( device->stym_table != NULL )
68         ( void ) lm_free ( ( char * ) device->stym_table ) ;
69 } /* free_stym_table */
70
71 void
72 free_device ( device )
73 struct device *device ;
74 /* frees the device and associated structures pointed to by device */
75 {
76     if ( device == NULL )
77         return ;
78     if ( device->device_name != NULL )
79         ( void ) lm_free ( device->device_name ) ;
80     if ( device->modeler_name != NULL )
81         ( void ) lm_free ( device->modeler_name ) ;
82     free_pin_table ( device ) ;
83     free_seq_table ( device ) ;
84     free_timing_table ( device ) ;
85     free_stym_table ( device ) ;
86     ( void ) lm_free ( ( char * ) device ) ;
87 } /* free_device */
88
89 /* end of routine to free a NML device */

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/print.c

DATE 5/23/89
TIME 4:42:24 pm

PAGE #
1/44

```

1  /* SCCS_ID: print.c rev 3.2, 5/9/89 at 15:25:06 */
2
3  /*
4   * Device Printer for BMDL Processor
5   */
6
7  #ifdef WRITE_DEVICE
8
9  #include "common.h"
10 #include "BMDL.h"
11 #include <strings.h>
12 #include "atrig.h"
13 #include "Bparse.h"
14 #include "trees.h"
15 #include "Bcost.h"
16
17 static void
18 print_attr ( attr )
19 struct pin_attr *attr;
20 /* prints the passed attribute */
21 {
22     switch ( attr->use_flag )
23     {
24         case ID_ATTR :
25             lm_print_message ( "%s", sytostr ( attr->attr.id ) );
26             break;
27         case CURRENT_SPEC :
28             switch ( attr->attr.cur_spec.cur )
29             {
30                 case C_1st :
31                     lm_print_message
32                     ( "%s = %s",
33                       sytostr ( attr->attr.cur_spec.cur ),
34                       attr->attr.cur_spec.val );
35                     break;
36                 case C_1st :
37                     lm_print_message
38                     ( "%s = %s",
39                       sytostr ( attr->attr.cur_spec.cur ),
40                       attr->attr.cur_spec.val );
41                     break;
42                 case C_pull_up :
43                 case C_pull_down :
44                     lm_print_message
45                     ( "%s = %s",
46                       sytostr ( attr->attr.cur_spec.cur ),
47                       attr->attr.cur_spec.val );
48                     break;
49                 default :
50                     break;
51             }
52             ASSERT(attr->attr.cur_spec.cur!=attr->attr.cur_spec.cur);
53             break;
54         case CYCLE_SPEC :
55             lm_print_message
56             ( "%s = %s",
57               sytostr ( attr->attr.cyc_spec.seq ),
58               sytostr ( attr->attr.cyc_spec.driv ) );
59             break;
60         default :
61             lm_error
62             ( 1,
63               /*LMSO*/"internal error: print_attr: bad use_flag: %d",
64               attr->use_flag );
65             break;
66     }
67 } /* print_attr */
68
69 static ushort
70 _valof ( sym )
71 symbol sym;
72 /* returns the integer value of the passed symbol */
73 {
74     char *str = sytostr ( sym );
75     do
76     {
77         val += 10;
78         val += *str - '0';
79     }
80     while ( ++str != '\0' );
81     return val;
82 } /* _valof */
83
84 static void
85 gen_sequence ( bits )
86 tree bits;
87 /* prints the source sequence that represents the passed tree */
88 {
89     if ( !bits )
90         return;
91     switch ( sameofnode ( bits ) )
92     {
93         case P_NUM :
94             lm_print_message ( "%s", sytostr ( textofnode ( bits ) ) );
95             break;
96         case N_LIST :
97             kids = kidcount ( bits );
98             gen_sequence ( subtree ( 1, bits ) );
99             for ( i = 2; i <= kids; ++i )
100                 lm_print_message ( "%s", sytostr ( textofnode ( subtree ( i, bits ) ) ) );
101             break;
102         case N_REPEAT :
103             kid = subtree ( 2, bits );
104             repeat = _valof ( textofnode ( subtree ( 1, bits ) ) );
105             lm_print_message ( "%d", repeat );
106             gen_sequence ( kid );
107             lm_print_message ( " " );
108             break;
109         case N_FEEDBACK :
110             lm_print_message
111             ( "%s",
112               sytostr ( textofnode ( subtree ( 1, bits ) ) ) );
113             break;
114     }
115     gen_sequence ( subtree ( 2, bits ) );
116 }

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/print.c	DATE 5/23/89	PAGE # 2/45
LINE #	SOURCE TEXT			
121	lm_print_message ("}");			
122	break ;			
123	default :			
124	lm_Error			
125	{			
126	/*MSG*/ "internal error: ba. in gen_sequence"			
127	}			
128	break ;			
129	}			
130	} /* gen_sequence */			
131				
132	static void			
133	print_pin_name (name)			
134	symbol name ;			
135	{ char *s ;			
136	s = syntostr (name) ;			
137	while (*s != '\0')			
138	{			
139	{ *s >= 'A' && *s <= 'Z'			
140	*s >= 'a' && *s <= 'z'			
141	*s >= '0' && *s <= '9'			
142	*s == '.' *s == '-' }			
143	{			
144	++s ;			
145	else			
146	break ;			
147	if (*s == '\0')			
148	{ lm_print_message ("s", syntostr (name)) ;			
149	return ;			
150	}			
151	lm_print_message ("") ;			
152	s = syntostr (name) ;			
153	while (*s != '\0')			
154	{ if (*s == '\n')			
155	lm_print_message ("") ;			
156	else			
157	lm_print_message ("c", *s) ;			
158	++s ;			
159	}			
160	lm_print_message ("") ;			
161	} /* print_pin_name */			
162				
163	static void			
164	print_sequence (seq)			
165	struct sequence *seq ;			
166	/* prints the passed sequence */			
167	{ lm_print_message (" ") ;			
168	print_pin_name (seq->pin_name) ;			
169	lm_print_message (" ") ;			
170	gen_sequence (seq->bits) ;			
171	lm_print_message ("\n") ;			
172	} /* print_sequence */			
173				
174	static ushort			
175	seen_pin (pins , i)			
176	struct pins *pins ;			
177	ushort i ;			
178	{ symbol name ;			
179	name = pins [i] . name ;			
180	while (i--)			
181	if (pins [i] . name == name)			
182	return i ;			
183	return 0 ;			
184	} /* seen_pin */			
185				
186	static void			
187	get_more_pins (pins , i , count)			
188	struct pins *pins ;			
189	ushort i , count ;			
190	{ symbol name ;			
191	name = pins [i] . name ;			
192	for (++i , i < count , ++i)			
193	if (pins [i] . name == name)			
194	{ lm_print_message (" ") ;			
195	print_pin_name (pins [i] . pkg_name) ;			
196	}			
197	} /* get_more_pins */			
198				
199	static void			
200	print_timing (min , typ , max)			
201	unsigned long min , typ , max ;			
202	{ ushort comma = 0 ;			
203	if (min != -1)			
204	{ lm_print_message ("min %g" , (float) min / 1000) ;			
205	comma = 1 ;			
206	}			
207	if (typ != -1)			
208	{ lm_print_message			
209	("%styp %g" , comma ? "," : "" , (float) typ / 1000) ;			
210	comma = 1 ;			
211	}			
212	if (max != -1)			
213	lm_print_message			
214	("%smax %g" , comma ? "," : "" , (float) max / 1000) ;			
215	} /* print_timing */			
216				
217	void			
218	print_dev (t)			
219	struct dev *t ;			
220	/* prints the passed device */			
221	{ ushort			
222	i ;			
223	{			
224	comma ;			
225	tree dir ;			
226	symbol num ;			
227	struct timing *tim ;			
228	if			
229	{ lm_fl_flag_on ()			
230	lm_lp_flag_on ()			
231	lm_rs_flag_on ()			
232	lm_rt_flag_on ()			
233	lm_rg_flag_on ()			
234	{			
235	lm_print_message ("\n") ;			
236	lm_print_message ("device_name ") ;			
237	print_pin_name (t->dev_name) ;			
238	lm_print_message ("\ndevice_usage %s\n" , syntostr (t->dev_type)) ;			
239	if (t->mod_name != NULLSTR)			
240	lm_print_message ("modeler_name %s\n" , syntostr (t->mod_name)) ;			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/print.c

DATE 5/23/89
TIME 4:42:24 pm

PAGE #
3/46

```

LINE # SOURCE TEXT
241 lm_print_message ( "clock_type %s\n", systostr ( t-> clk_type ) );
242 lm_print_message ( "device speed\n" );
243 if ( t-> clk_period1 & 1000000 == 0 )
244 lm_print_message ( "tg us - ", ( float ) t-> clk_period1 / 1000000 );
245 else
246 lm_print_message ( "tg ns - ", ( float ) t-> clk_period1 / 1000 );
247 if ( t-> clk_period2 == 0 )
248 lm_print_message ( "infinity\n" );
249 else if ( t-> clk_period2 & 1000000 == 0 )
250 lm_print_message ( "tg us\n", ( float ) t-> clk_period2 / 1000000 );
251 else
252 lm_print_message ( "tg ns\n", ( float ) t-> clk_period2 / 1000 );
253 if ( t-> ultra_fast )
254 lm_print_message ( "ultra_fast\n" );
255 if ( t-> dis_tlm_check )
256 lm_print_message ( "disable_timing_checking\n" );
257 if ( t-> inh_tlm_measure )
258 lm_print_message ( "inhibit_timing_measurement\n" );
259 if ( t-> missing_delays != NULLSYM )
260 lm_print_message ( "report_missing_delays %s\n",
261 systostr ( t-> missing_delays ) );
262 if ( t-> io_store_changes != NULLSYM )
263 lm_print_message ( "report_io_store_changes %s\n",
264 systostr ( t-> io_store_changes ) );
265 lm_print_message ( "technology %s", systostr ( t-> technology ) );
266 lm_print_message ( "vcc= %s", t-> Vcc );
267 lm_print_message ( "vth= %s", t-> Vth );
268 lm_print_message ( "vlim= %s", t-> Vlim );
269 lm_print_message ( "vil= %s", t-> Vil );
270 lm_print_message ( "voh= %s", t-> Voh );
271 lm_print_message ( "val= %s", t-> Val );
272 lm_print_message ( "iil= %s", t-> Iil );
273 lm_print_message ( "iib= %s", t-> Iib );
274 lm_print_message ( "iol= %s", t-> Iol );
275 lm_print_message ( "iob= %s", t-> Iob );
276 if ( t-> def_setup != UNDEFINED )
277 lm_print_message ( "device_setup_time %s\n", t-> def_setup / 1000 );
278 if ( t-> def_hold != UNDEFINED )
279 lm_print_message ( "device_hold_time %s\n", t-> def_hold / 1000 );
280 if ( t-> def_sample != UNDEFINED )
281 lm_print_message ( "device_sample_time %s\n", t-> def_sample / 1000 );
282 for ( i = 0; i < t-> pin_cnt; ++i )
283 if ( t-> pins [ i ]. number < MAX_PINS )
284 {
285     sum = t-> pins [ i ]. name;
286     dir = t-> pins [ i ]. dir;
287     if ( nameofnode ( dir ) == M_POWER_PINS )
288     {
289         if ( ! sees_pin ( t-> pins, i ) )
290             lm_print_message ( "power_pin - " );
291         print_pin_name ( sum );
292         lm_print_message ( " = " );
293         print_pin_name ( t-> pins [ i ]. pkg_name );
294         get_more_pins ( t-> pins, i, t-> pin_cnt );
295         lm_print_message ( "\n" );
296     }
297     else if ( nameofnode ( dir ) == M_GROUND_PINS )
298     {
299         if ( ! sees_pin ( t-> pins, i ) )
300             lm_print_message ( "ground_pin - " );
301         print_pin_name ( sum );
302         lm_print_message ( " = " );
303         print_pin_name ( t-> pins [ i ]. pkg_name );
304         get_more_pins ( t-> pins, i, t-> pin_cnt );
305         lm_print_message ( "\n" );
306     }
307     else if ( nameofnode ( dir ) == M_NC_PINS )
308     {
309         if ( ! sees_pin ( t-> pins, i ) )
310             lm_print_message ( "nc_pin - " );
311         print_pin_name ( sum );
312         lm_print_message ( " = " );
313         print_pin_name ( t-> pins [ i ]. pkg_name );
314         get_more_pins ( t-> pins, i, t-> pin_cnt );
315         lm_print_message ( "\n" );
316     }
317     else
318     {
319         ASSERT ( ! pin_name_of ( sum ) );
320         lm_print_message ( "in_pin - " );
321         nameofnode ( dir ) == M_IN_PINS ? "in_pin " :
322         "out_pin "
323         print_pin_name ( sum );
324         lm_print_message ( " = " );
325         print_pin_name ( t-> pins [ i ]. pkg_name );
326         if ( t-> pins [ i ]. sum_attr )
327         {
328             comma = 0;
329             for ( j = 0; j < t-> pins [ i ]. sum_attr; ++j )
330             {
331                 lm_print_message ( " %c", comma ? "," : "" );
332                 print_attr ( t-> pins [ i ]. attr + j );
333                 comma = 1;
334             }
335             lm_print_message ( "\n" );
336         }
337     }
338 }
339 lm_print_message ( "package_mappings\n" );
340 for ( i = 0; i < t-> pin_cnt; ++i )
341 if ( t-> pins [ i ]. number < MAX_PINS )
342 {
343     print_pin_name ( t-> pins [ i ]. pkg_name );
344     lm_print_message ( " %s\n", t-> pins [ i ]. number );
345 }
346 lm_print_message ( "adapter_mappings\n" );
347 for ( i = 0; i < t-> pin_cnt; ++i )
348 if ( t-> pins [ i ]. number < MAX_PINS )
349 {
350     lm_print_message ( " %s\n", t-> pins [ i ]. number );
351     lm_print_message ( " %s\n", t-> pins [ i ]. number );
352     if ( t-> pins [ i ]. trace_delay )
353     {
354         lm_print_message ( " %s\n", t-> pins [ i ]. trace_delay );
355     }
356 }
357

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/print.c	DATE 5/23/89	PAGE # 4/47
LINE #	SOURCE TEXT			
361	<pre> (float) t -> pins [i] . trace_delay / 1000 } lm_print_message ("\n"); if (t -> pins [i] . alias != NULLSYM) j = 1; } if (j) { lm_print_message ("pin_name_mapping\n"); for (i = 0; i < t -> pin_cat; ++i) { if (t -> pins [i] . number < MAX_PINS) { if (t -> pins [i] . alias != NULLSYM) { print_pin_name (t -> pins [i] . alias); lm_print_message ("\n"); print_pin_name (t -> pins [i] . name); lm_print_message ("\n"); } } } } if (t -> num_seqs) { lm_print_message ("initialize\n"); for (i = 0; i < t -> num_seqs; ++i) print_sequence (t -> sequences + i); } if (t -> use_default) { lm_print_message ("default_delay\n"); print_timing (t -> min_default, t -> typ_default, t -> max_default); lm_print_message ("\n"); } for (i = 0; i < t -> pin_cat; ++i) { tim = t -> pins [i] . timing_list; while (tim != NULL) { lm_print_message ("delay from %s(%s)\n", systr (tim -> minor_state), print_pin_name (tim -> minor_pin_name)); lm_print_message ("/*(%) to %s(%s)\n", systr (tim -> major_state), print_pin_name (t -> pins [i] . name)); lm_print_message ("/*(%)*/\n"); lm_print_message ("\n"); print_timing (tim -> minimum, tim -> typical, tim -> maximum); lm_print_message ("\n"); tim = tim -> next; } } } /* print_dev */ static void print_pin (T) tree T; { if (sameofnode (T) == N_PIN_NAME) { lm_print_message (systr (textofnode (subtree (1, T)))); lm_print_message ("[-"); lm_print_message (systr (textofnode (subtree (2, T)))); if (kidcount (T) == 3) { lm_print_message (" "); lm_print_message (systr (textofnode (subtree (3, T)))); } lm_print_message ("]"); } else { lm_print_message (systr (textofnode (T))); } } /* print_pin */ static void print_pin_map (T) tree T; { ushort i, kids; tree k; kids = kidcount (T); for (i = 1; i <= kids; ++i) { k = subtree (i, T); for (j = 1; sameofnode (k = subtree (j, k)) != N_NAMES; ++j) { lm_print_message (" "); print_pin (k); lm_print_message ("\t"); print_pin (k); lm_print_message (" \\\n"); } } } /* print_pin_map */ void print_map (T) tree T; { ushort i, kids; lm_print_message ("(If you change the pin name mapping "); lm_print_message ("of any of the pins defined bel. \\\n"); lm_print_message ("(you must uncomment the following "); lm_print_message ("line by removing the two braces.\\n"); lm_print_message ("pin_name_mapping\n"); lm_print_message ("\n(The following lines define the "); lm_print_message ("pin name mapping for this device.\\n"); lm_print_message ("\n(Change the pin name on the left "); lm_print_message ("of the equals to correspond with your\\n"); lm_print_message ("(CAE system's pin names. Do not "); lm_print_message ("change the name on the right of the\\n"); lm_print_message ("(equals. If the pin name already "); lm_print_message ("matches your CAE system's pin name.\\n"); lm_print_message ("(I do not change the pin name and don't "); lm_print_message ("remove the braces from the line.\\n"); kids = kidcount (T); lm_print_message ("\n(input pins (in_pin) \\\n"); for (i = 1; i <= kids; ++i) { k = subtree (i, T); if (sameofnode (k) == N_IN_PINS) print_pin_map (k); } lm_print_message ("\n(output pins (out_pin) \\\n"); </pre>			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/print.c

DATE 5/23/89
TIME 4:42:24 pm

PAGE #
5/48

```

LINE # SOURCE TEXT
481 for ( i = 1, i <= kids, ++i )
482 { kid = subtree ( i, T );
483 if ( nameofnode ( kid ) == N_OUT_PINS )
484 print_pin_map ( kid );
485 }
486 lm_print_message ( "\n bidirectional pins (io_pin) \n" );
487 for ( i = 1, i <= kids, ++i )
488 { kid = subtree ( i, T );
489 if ( nameofnode ( kid ) == N_IO_PINS )
490 print_pin_map ( kid );
491 }
492 lm_print_message ( "\n power pins (power_pin) \n" );
493 for ( i = 1, i <= kids, ++i )
494 { kid = subtree ( i, T );
495 if ( nameofnode ( kid ) == N_POWER_PINS )
496 print_pin_map ( kid );
497 }
498 lm_print_message ( "\n ground pins (ground_pin) \n" );
499 for ( i = 1, i <= kids, ++i )
500 { kid = subtree ( i, T );
501 if ( nameofnode ( kid ) == N_GROUND_PINS )
502 print_pin_map ( kid );
503 }
504 lm_print_message ( "\n no-connect pins (nc_pin) \n" );
505 for ( i = 1, i <= kids, ++i )
506 { kid = subtree ( i, T );
507 if ( nameofnode ( kid ) == N_NC_PINS )
508 print_pin_map ( kid );
509 }
510 } /* print_map */
511
512 static void
513 dump_pin ( T, pin )
514 tree T,
515 ushort pin,
516 { ushort p, names, j, k, i, kids, comma = 0;
517 tree names1, names2, kid;
518 symbol s;
519 char num [ 12 ];
520 ( void ) sprintf ( num, "%u", pin );
521 s = strtosym ( num );
522 kids = kidcount ( T );
523 for ( i = 1, i <= kids, ++i )
524 { kid = subtree ( i, T );
525 if ( nameofnode ( kid ) == N_ADAPTER )
526 { k = kidcount ( kid );
527 for ( j = 1, j <= k, ++j )
528 { names1 = subtree ( 1, subtree ( j, kid ) );
529 names2 = subtree ( 2, subtree ( j, kid ) );
530 names = kidcount ( names2 );
531 ASSERT(kidcount(names1)==names);
532 for ( p = 1, p <= names, ++p )
533 if ( textofnode ( subtree ( p, names2 ) ) == n )
534 { lm_print_message
535 ( "test",
536 comma ? ", " : "",
537 symsostr ( textofnode ( subtree ( p, names1 ) ) ) );
538 }
539 comma = 1;
540 }
541 }
542 }
543 lm_print_message ( " " );
544 } /* dump_pin */
545
546 void
547 dump_power_ground ( t, T )
548 struct dev *t,
549 tree T,
550 { ushort i;
551 symbol name;
552 lm_print_message ( "VCC\n" );
553 for ( i = 0, i < t->pin_cnt, ++i )
554 if ( t->pins [ i ]. number < MAX_PINS )
555 { name = t->pins [ i ]. name;
556 if ( nameofnode ( g_lookup ( name ) ) == N_POWER_PINS )
557 dump_pin ( T, t->pins [ i ]. number );
558 }
559 lm_print_message ( "\nGND\n" );
560 for ( i = 0, i < t->pin_cnt, ++i )
561 if ( t->pins [ i ]. number < MAX_PINS )
562 { name = t->pins [ i ]. name;
563 if ( nameofnode ( g_lookup ( name ) ) == N_GROUND_PINS )
564 dump_pin ( T, t->pins [ i ]. number );
565 }
566 lm_print_message ( "\n" );
567 } /* dump_power_ground */
568
569 #endif WRITE_DEVICE
570
571 /* end of Device Printer for BNPL Processor */
572
573

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/strng.c

DATE 5/23/89 PAGE #
TIME 4:42:25 pm 1/49

```

1  /* SCCS_ID: strng.c rev 3.2, 5/9/89 at 15:20:30 */
2
3  /*
4   * String Module
5   */
6
7  #include <stdio.h>
8  #include <strings.h>
9  #include "strng.h"
10
11 #define NOMEM 1 /* no more memory error code */
12 #define NOSYMS 2 /* no more symbol space error code */
13 #define NOTEXT 3 /* no more text space error code */
14 #define BUCKETS 256 /* number of hash buckets */
15 #define MAXSYMS 65535
16 #define SYMHEADSIZE 256
17 #define SYMLOCKSIZE 256
18 #define SYMHEAD(S) (S >> 8)
19 #define SYMLOCK(S) (S & 255)
20 #define TEXTHEADSIZE 64
21 #define TEXTLOCKSIZE 1024
22 #define TEXTHEAD(T) (T >> 10)
23 #define TEXTLOCK(T) (T & 1023)
24
25 typedef unsigned short tree; /* ASSUMES! that a tree is an unsigned short */
26
27 extern void
28   lm_free ( ),
29   lm_print_message ( ),
30   lm_error ( ), /* to report an error */
31   lm_fatal ( ), /* to cause an exit */
32
33 extern char *lm_malloc ( );
34
35 static struct symbol /* symbol structure */
36 {
37   unsigned short systext; /* index of the text of the symbol */
38   symbol nextsym; /* index of next symbol in bucket */
39   unsigned short
40     pin_name, /* device signal name */
41     pin_number, /* package pin number */
42     apin_name, /* adapter signal name */
43     apin_number, /* adapter pin number */
44     tree;
45   /* global declaration */
46   /* local declaration */
47   symbol *symlist; /* the 0'th symbol block */
48   /* array of pointers to symbol structures */
49   static symbol
50     hashbuckets [ BUCKETS ], /* indices into symspace */
51     lastsym; /* last symbol */
52   static char textlock0 [ TEXTLOCKSIZE ]; /* the 0'th text block */
53   static char *textheader [ TEXTHEADSIZE ]; /* array of pointers to text blocks */
54   static unsigned short
55     lastchar,
56     len; /* index of last character, length of current string */
57 }
58
59 void
60 strinit ( )
61 {
62   register unsigned short i;
63   symlist = symlist0; /* 0'th symbol block is symlist0 */
64   for ( i = 1; i < SYMHEADSIZE; ++i )
65     symlist [ i ] = NULL; /* all other symbol blocks unallocated */
66   textheader = textlock0; /* 0'th text block is textlock0 */
67   for ( i = 1; i < TEXTHEADSIZE; ++i )
68     textheader [ i ] = NULL; /* all other text blocks unallocated */
69   for ( i = 0; i < BUCKETS; ++i )
70     hashbuckets [ i ] = NULLSYM; /* all buckets are empty */
71   symlist0 = systext = 0; /* empty text */
72   textlock0 = "0"; /* null symbol is empty */
73   lastsym = NULLSYM;
74   lastchar = 0;
75 } /* strinit */
76
77 void
78 strfree ( )
79 {
80   register unsigned short i;
81   for ( i = 1; i < SYMHEADSIZE; ++i )
82     if ( symlist [ i ] != NULL )
83       lm_free ( (char *) symlist [ i ] );
84   for ( i = 1; i < TEXTHEADSIZE; ++i )
85     if ( textheader [ i ] != NULL )
86       lm_free ( textheader [ i ] );
87 } /* strfree */
88
89 static void
90 strerr ( err )
91 register unsigned short err;
92 {
93   switch ( err )
94   {
95     case NOMEM:
96       lm_error ( 0 );
97       /*LMSC*/"out of memory on modeler (strings)"
98     }
99   break;
100   case NOSYMS:
101     lm_error ( 0 );
102     /*LMSC*/"internal error: no more symbol space (too many symb's)"
103   }
104   break;
105   case NOTEXT:
106     lm_error ( 0 );
107     /*LMSC*/"internal error: no more text space (too much symbol text)"
108   }
109   break;
110   default:
111     lm_error ( 0 );
112     /*LMSC*/"internal error: unknown string module error"
113   }
114   break;
115   lm_fatal ( );
116 } /* strerr */
117
118 static unsigned short
119 hash ( str )
120 register char *str;
121 {
122   unsigned char ret;

```


Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/strng.c

DATE 5/23/89

PAGE #

TIME 4:42:25 pm

2/50

```

121 len = strlen ( str ) ; /* note side-effect */
122 switch ( len )
123 {
124     case 1 :
125         ret = ( *str << 1 ) + 1 ;
126         break ;
127     case 2 :
128         ret = ( *str << 1 ) ^ str [ 1 ] ;
129         break ;
130     default :
131         ret =
132             ( len & 0xFF ) ^
133             *str
134             ( str [ len >> 1 ] << 1 ) ^
135             str [ len - 1 ] ;
136         break ;
137 }
138 return ret ;
139 /* hash */
140
141 symbol
142 strtosym ( str )
143 register char *str ;
144 {
145     register symbol i ;
146     register char *text ;
147     register struct symbol *sym ;
148     register unsigned short
149         h ,
150         ndex ,
151         empty ;
152     if ( *str == '\0' )
153         return NULLSYM ;
154     h = hash ( str ) ;
155     if ( hashbuckets [ h ] == NULLSYM )
156         empty = 1 ;
157     else
158         {
159             i = hashbuckets [ h ] ;
160             do
161             {
162                 sym = symheader [ SYNHEAD ( i ) ] + SYMBLOCK ( i ) ;
163                 ndex = sym -> syntext ;
164                 if
165                     ( strcmp
166                     ( str ,
167                     textheader [ TEXTHEAD ( ndex ) ] + TEXTBLOCK ( ndex )
168                     ) == 0 )
169                     return i ;
170             }
171             while ( i != NULLSYM ) ;
172             empty = 0 ;
173         }
174     if ( lastsym == MAXSYM )
175         strerr ( NOSYM ) ;
176     if ( empty )
177         hashbuckets [ h ] = ++lastsym ;
178     else
179         sym -> nextsym = ++lastsym ;
180     ndex = SYNHEAD ( lastsym ) ;
181     if ( symheader [ ndex ] == NULL )
182         {
183             sym =
184                 ( struct symbol * ) malloc
185                 ( sizeof ( struct symbol ) * SYMBLOCKSIZE ) ;
186             if ( sym == NULL )
187                 strerr ( NOMEM ) ;
188             symheader [ ndex ] = sym ;
189             /* this could be one line */
190             sym -> nextsym = NULLSYM ;
191             sym -> nextsym = NULLSYM ;
192             sym -> pin_name =
193             sym -> pin_number =
194             sym -> spin_name =
195             sym -> spin_number =
196             sym -> g_dcls =
197             sym -> l_dcls =
198             0 ;
199             i = TEXTBLOCK ( ++lastchar ) ;
200             if ( i + len > TEXTBLOCKSIZE )
201                 {
202                     lastchar += TEXTBLOCKSIZE - 1 ; /* wastage */
203                     if ( lastchar == 0 )
204                         strerr ( NOTEXT ) ;
205                     i = 0 ;
206                 }
207             sym -> syntext = lastchar ;
208             ndex = TEXTHEAD ( lastchar ) ;
209             if ( textheader [ ndex ] == NULL )
210                 {
211                     text = malloc ( TEXTBLOCKSIZE ) ;
212                     if ( text == NULL )
213                         strerr ( NOMEM ) ;
214                     textheader [ ndex ] = text ;
215                     /* this could be one line */
216                     text = textheader [ ndex ] + 1 ;
217                     do
218                     {
219                         ++lastchar ;
220                         *text++ = *str++ ;
221                     }
222                     while ( *str ) ;
223                     *text = '\0' ;
224                     return lastsym ;
225                 }
226             /* strtosym */
227
228 symbol
229 strtolook ( str )
230 register char *str ;
231 {
232     register symbol i ;
233     register struct symbol *sym ;
234     register unsigned short ndex ;
235     if ( *str == '\0' )
236         return NULLSYM ;
237     if ( i = hashbuckets [ hash ( str ) ] != NULLSYM )
238         do
239         {
240             sym = symheader [ SYNHEAD ( i ) ] + SYMBLOCK ( i ) ;
241             ndex = sym -> syntext ;
242             if
243                 ( strcmp
244                 ( str ,
245                 textheader [ TEXTHEAD ( ndex ) ] + TEXTBLOCK ( ndex )

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/strng.c

DATE 5/23/89
TIME 4:42:25 pm

PAGE #
3/51

```

LINE # SOURCE TEXT
241     } == 0
242     }
243     return i ;
244     else
245     {
246         i = sym -> nextsym ;
247         while ( i != NULLSYM )
248             return NULLSYM ;
249     } /* strlock */
250
251     /*symbol
252     strlast ( )
253     {
254         return lastsym ;
255     } strlast */
256
257     char *
258     symtostr ( sym )
259     {
260         register symbol sym ;
261         register unsigned short i ;
262         i = symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . symtext ;
263         return textheader [ TEXTHEAD ( i ) ] + TEXTBLOCK ( i ) ;
264     } /* symtostr */
265
266     void
267     set_pin_name ( sym , name )
268     {
269         register symbol sym ;
270         register unsigned short name ;
271         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . pin_name = name ;
272     } /* set_pin_name */
273
274     void
275     set_pin_number ( sym , num )
276     {
277         register symbol sym ;
278         register unsigned short num ;
279         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . pin_number = num ;
280     } /* set_pin_number */
281
282     void
283     set_apin_name ( sym , name )
284     {
285         register symbol sym ;
286         register unsigned short name ;
287         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . apin_name = name ;
288     } /* set_apin_name */
289
290     void
291     set_apin_number ( sym , num )
292     {
293         register symbol sym ;
294         register unsigned short num ;
295         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . apin_number = num ;
296     } /* set_apin_number */
297
298     void
299     g_declare ( sym , T )
300     {
301         register symbol sym ;
302         register tree T ;
303         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . g_dcls = T ;
304     } /* g_declare */
305
306     void
307     l_declare ( sym , T )
308     {
309         register symbol sym ;
310         register tree T ;
311         symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . l_dcls = T ;
312     } /* l_declare */
313
314     unsigned short
315     pin_name_of ( sym )
316     {
317         register symbol sym ;
318         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . pin_name ;
319     } /* pin_name_of */
320
321     unsigned short
322     pin_number_of ( sym )
323     {
324         register symbol sym ;
325         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . pin_number ;
326     } /* pin_number_of */
327
328     unsigned short
329     apin_name_of ( sym )
330     {
331         register symbol sym ;
332         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . apin_name ;
333     } /* apin_name_of */
334
335     unsigned short
336     apin_number_of ( sym )
337     {
338         register symbol sym ;
339         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . apin_number ;
340     } /* apin_number_of */
341
342     tree
343     g_lookup ( sym )
344     {
345         register symbol sym ;
346         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . g_dcls ;
347     } /* g_lookup */
348
349     tree
350     l_lookup ( sym )
351     {
352         register symbol sym ;
353         return symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . l_dcls ;
354     } /* l_lookup */
355
356     /*tree
357     lookup ( sym )
358     {
359         register symbol sym ;
360         register tree dcls ;
361         return
362             ( dcls = symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . l_dcls ) ?
363             dcls :
364             symheader [ SYMHEAD ( sym ) ] [ SYMBLOCK ( sym ) ] . g_dcls ;
365     } lookup */
366
367     #ifdef STATS
368     void
369     strstat ( )
370     {
371         register symbol j ;
372         register unsigned short
373             i ,
374             buckets ,
375             onelong ,
376             longest ;

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM
shellswm/strng.c

DATE 5/23/89

PAGE #

TIME 4:42:25 pm

4/52

```

LINE # SOURCE TEXT
361     longs ,
362     length ,
363     buckets =
364     onelong =
365     longest =
366     0
367     lm_print_message
368     ( "td unique symbols, td total characters\n" ,
369     lastsym ,
370     lastsym == 1 ? "" : "s" ,
371     lastchar - lastsym ,
372     lastchar - lastsym == 1 ? "" : "s"
373     ) ,
374     for ( i = 0 ; i < BUCKETS ; ++i )
375     if ( hashbuckets [ i ] != NULLSYM )
376     {
377         ++buckets ;
378         j = hashbuckets [ i ] ;
379         length = 0 ;
380         do
381         {
382             ++length ;
383             j = symheader [ SYMHEAD ( j ) ] [ SYMBLOCK ( j ) ] . nextsym ;
384         } while ( j != NULLSYM ) ;
385         if ( length == 1 )
386             ++onelong ;
387         if ( length > longest )
388             longest = length ;
389         longs = 1 ;
390     }
391     else if ( length == longest )
392         ++longs ;
393     lm_print_message
394     ( "td Buckets used" ,
395     buckets ,
396     buckets == 1 ? "" : "s"
397     ) ,
398     if ( buckets )
399     lm_print_message ( " , td of length 1" , onelong ) ,
400     if ( longest > 1 )
401     lm_print_message ( " , td of length td" , longs , longest ) ,
402     lm_print_message ( "\n" ) ;
403     /* strstat */
404
405     /*void
406     strdump ( )
407     { register symbol j ;
408     register unsigned short i ;
409     lm_print_message ( "buckets:\n" ) ;
410     for ( i = 0 ; i < BUCKETS ; ++i )
411     if ( hashbuckets [ i ] != NULLSYM )
412     { j = hashbuckets [ i ] ;
413     lm_print_message ( "td:td" , i , j ) ;
414     while
415     {
416         i = symheader [ SYMHEAD ( j ) ] [ SYMBLOCK ( j ) ] . nextsym ;
417         lm_print_message ( "td" , j ) ;
418         lm_print_message ( "\n" ) ;
419     }
420     }
421     lm_print_message ( "strings:\n" ) ;
422     for ( j = 1 ; j <= lastsym ; ++j )
423     lm_print_message ( "td:td\n" , j , syntostr ( j ) ) ;
424     } strdump */
425     #endif STATE
426
427     /* end of String Module */
428

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/trees.c	DATE 5/23/89	PAGE # 1/53
LINE #	SOURCE TEXT			
1	/* SCCS_ID: trees.c rev 3.2, 5/9/89 at 15:22:40 */			
2	/*			
3	Tree Module			
4	*/			
5	#include <stdio.h>			
6	#include <strings.h>			
7	#include "string.h"			
8	#include "trees.h"			
9				
10				
11				
12	#define NOMEM 1 /* no more memory error code */			
13	#define NOMEMES 2 /* no more node space error code */			
14	#define NODEHEADSIZE 256			
15	#define NODEBLOCKSIZE 256			
16	#define NODEHEAD(N)(N>8)			
17	#define NODEBLOCK(N)(N&255)			
18				
19	extern void			
20	lm_free ()			
21	lm_print_message ()			
22	lm_error () /* to report an error */			
23	lm_fatal () /* to cause an exit */			
24				
25	extern char *lm_malloc ()			
26				
27	extern unsigned short lm_get_lineno ()			
28				
29	extern symbol source			
30				
31	static struct node nodeblock0 [NODEBLOCKSIZE] /* 0'th node block */			
32	static struct node *nodeheader [NODEHEADSIZE] /* node space header */			
33	static tree			
34	last_tree			
35	tree_top /* last tree, top of stack */			
36				
37	void			
38	treeinit ()			
39	{ register unsigned short i /* point to the 0'th node block */			
40	nodeblock0 = nodeblock0 /* point to the 0'th node block */			
41	for (i = 1 ; i < NODEHEADSIZE ; ++i)			
42	nodeheader [i] = NULL /* the rest of the blocks are unallocated */			
43	last_tree = NULLTREE /* skip the 0'th tree */			
44	nodeblock0->name =			
45	nodeblock0->arc =			
46	NULLSTR			
47	nodeblock0->kids =			
48	nodeblock0->first =			
49	nodeblock0->arcline =			
50	0			
51	#ifdef DECOS			
52	nodeblock0->decor = 0			
53	#endif DECOS			
54	tree_top = 0 /* first push will use the 45335'th spot */			
55	/* treeinit */			
56				
57	void			
58	treefree ()			
59	{ register unsigned short i			
60	for (i = 1 ; i < NODEHEADSIZE ; ++i)			
61	if (nodeheader [i] != NULL)			
62	(void) lm_free ((char *) nodeheader [i])			
63	/* treefree */			
64				
65	static void			
66	treeerr (err)			
67	register unsigned short err			
68	{ switch (err)			
69	{ case NOMEM :			
70	lm_error			
71	{			
72	/*MSG*/"out of memory on modeler (trees)"			
73	break			
74	case NOMEMES :			
75	{			
76	lm_error			
77	{			
78	/*MSG*/"internal error: no more node space (specification too large)"			
79	break			
80	default :			
81	lm_error			
82	{			
83	/*MSG*/"internal error: unknown tree module error"			
84	break			
85	}			
86	}			
87	lm_fatal ()			
88	/* treeerr */			
89				
90				
91	void			
92	bldtrn (name , text)			
93	register symbol			
94	name			
95	text			
96	{ register struct node *node			
97	register unsigned short i			
98	if (last_tree == tree_top - 1)			
99	trees = NOMEMES			
100	tree_top =			
101	i = NODEHEAD (tree_top)			
102	if (nodeheader [i] == NULL)			
103	{ node =			
104	{ struct node *) lm_malloc			
105	{ sizeof (struct node) * NODEBLOCKSIZE			
106	}			
107	if (node == NULL)			
108	treeerr (NOMEM)			
109	nodeheader [i] = node			
110	/* this could be one line */			
111	node = nodeheader [i] + NODEBLOCK (tree_top)			
112	node->name = name			
113	node->first = text			
114	node->kids = 0			
115	node->arc = source			
116	node->arcline = lm_get_lineno ()			
117	/* bldtrn */			
118				
119	void			
120	bldostrn (name , kids)			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM
shellswm/trees.c

DATE 5/23/89

PAGE #

TIME 4:42:25 pm

2/54

```

LINE # SOURCE TEXT
121 register symbol name ;
122 register unsigned short kids ;
123 { register unsigned short i , adex ;
124 register struct node *node , *top ;
125 if ( last_tree == tree_top - 1 )
126 treeerr ( MOMEM ) ;
127 for ( i = 0 , i < kids , ++i )
128 { ++last_tree ;
129 adex = NODEHEAD ( last_tree ) ;
130 if ( nodeheader [ adex ] == NULL )
131 { node =
132 ( struct node * ) malloc
133 ( sizeof ( struct node ) * NODEBLOCKSIZE ) ;
134 }
135 if ( node == NULL )
136 treeerr ( MOMEM ) ;
137 nodeheader [ adex ] = node ;
138 /* this could be one line */
139 node = nodeheader [ adex ] + NODEBLOCK ( last_tree ) ;
140 top = nodeheader [ NODEHEAD ( tree_top ) ] + NODEBLOCK ( tree_top ) ;
141 node->name = top->name ;
142 node->kids = top->kids ;
143 node->src = top->src ;
144 node->arcline = top->arcline ;
145 node->first = top->first ;
146 #ifdef DECOR
147 node->decor = 0 ;
148 #endif DECOR
149 ++tree_top ;
150 }
151 --tree_top ;
152 adex = NODEHEAD ( tree_top ) ;
153 if ( nodeheader [ adex ] == NULL )
154 { node =
155 ( struct node * ) malloc
156 ( sizeof ( struct node ) * NODEBLOCKSIZE ) ;
157 }
158 if ( node == NULL )
159 treeerr ( MOMEM ) ;
160 nodeheader [ adex ] = node ;
161 /* this could be one line */
162 node = nodeheader [ adex ] + NODEBLOCK ( tree_top ) ;
163 node->name = name ;
164 node->kids = kids ;
165 node->first = i == 0 ? NULLSYM : last_tree ;
166 node->src = source ;
167 node->arcline = la_get_arcline ( ) ;
168 } /* kidcount */
169
170 tree
171 pop_top ( )
172 { register unsigned short i ;
173 register struct node *node , *top ;
174 if ( tree_top == 0 )
175 return NULLTREE ;
176 ++last_tree ;
177 i = NODEHEAD ( last_tree ) ;
178 if ( nodeheader [ i ] == NULL )
179 { node =
180 ( struct node * ) malloc
181 ( sizeof ( struct node ) * NODEBLOCKSIZE ) ;
182 }
183 if ( node == NULL )
184 treeerr ( MOMEM ) ;
185 nodeheader [ i ] = node ;
186 /* this could be one line */
187 node = nodeheader [ i ] + NODEBLOCK ( last_tree ) ;
188 top = nodeheader [ NODEHEAD ( tree_top ) ] + NODEBLOCK ( tree_top ) ;
189 node->name = top->name ;
190 node->kids = top->kids ;
191 node->src = top->src ;
192 node->arcline = top->arcline ;
193 node->first = top->first ;
194 #ifdef DECOR
195 node->decor = 0 ;
196 #endif DECOR
197 ++tree_top ;
198 return last_tree ;
199 } /* pop_top */
200
201 symbol
202 nameofnode ( T )
203 { register tree T ;
204 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . name ;
205 } /* nameofnode */
206
207 symbol
208 textofnode ( T )
209 { register tree T ;
210 { return
211 nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . kids >
212 NULLSYM :
213 nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first ;
214 } /* textofnode */
215
216 void
217 set_text ( T , text )
218 { register tree T ;
219 register symbol text ;
220 { nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first = text ;
221 } /* set_text */
222
223 unsigned short
224 kidcount ( T )
225 { register tree T ;
226 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . kids ;
227 } /* kidcount */
228
229 tree
230 subtree ( n , T )
231 { register unsigned short n ;
232 register tree T ;
233 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first + 1 - n ;
234 } /* subtree */
235
236 /* unsigned short
237 whichkid ( T , kid )
238 register tree
239 T
240 kid ;

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/trees.c

DATE

5/23/89

PAGE #

TIME

4:42:25 pm

3/55

```

LINE # SOURCE TEXT
241 { return
242 nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first - kid + 1 ,
243 } whichkid */
244
245 symbol
246 file_of ( T )
247 register tree T ,
248 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . arc ,
249 } /* file_of */
250
251 symbol
252 line_of ( T )
253 register tree T ,
254 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . arcline ,
255 } /* line_of */
256
257 #ifdef DECOR
258 void
259 decorate ( T , dec )
260 register tree T ,
261 register long dec ,
262 { nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . decor = dec ,
263 } /* decorate */
264
265 long
266 decoration ( T )
267 register tree T ,
268 { return nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . decor ,
269 } /* decoration */
270 #endif DECOR
271
272 static void
273 tprint ( T , depth )
274 register tree T ,
275 register unsigned short depth ,
276 { register unsigned short i , kids ,
277   for ( i = 0 , i < depth , ++i )
278     lm_print_message ( " " ) ,
279     lm_print_message ( "\n" ) ,
280     kids = nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . kids ,
281     ++depth ,
282     for ( i = 1 , i <= kids , ++i )
283       tprint (
284         nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first + i - 1 ,
285         depth
286       )
287   } /* tprint */
288
289 void
290 printtree ( T )
291 register tree T ,
292 { tprint ( T , 0 ) ,
293 } /* printtree */
294
295 void
296 printnode ( T )
297 register tree T ,
298 { register unsigned short kids ,
299   #ifdef DECOR
300     tree dec ,
301     #endif DECOR
302   if ( T == NULLTREE )
303     { lm_print_message ( "NULLTREE" ) ,
304       return ,
305     }
306   if
307     { ( kids = nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . kids ) ||
308       nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first == NULLSYM
309     }
310   lm_print_message
311     { "ts[0u] (0u kids)" ,
312     }
313   systostr
314     { nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . name
315     } ,
316   T ,
317   kids ,
318   kids == 1 ? "" : "s"
319   )
320   else
321     lm_print_message
322     { "ts[0u] : ts" ,
323     }
324     { nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . name
325     } ,
326     T ,
327     systostr
328     { nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . first
329     }
330     )
331   #ifdef DECOR
332     if ( dec = nodeheader [ NODEHEAD ( T ) ] [ NODEBLOCK ( T ) ] . decor )
333       lm_print_message ( " : id" , dec ) ,
334     #endif DECOR
335   } /* printnode */
336
337 /* end of tree module */
338

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE 5/23/89
TIME 4:42:25 pm

PAGE #
1/56

LINE # SOURCE TEXT

```

1  /* SCCS ID: walk_1.c rev 3.1, 4/24/89 at 08:01:37 */
2
3  /*
4   * Constrainer (walk_1) for EMUL Processor
5   */
6
7  #include "common.h"
8  #include "EMUL.h"
9  #include "strings.h"
10 #include "strng.h"
11 #include "Sparse.h"
12 #include "trees.h"
13 #include "Econst.h"
14
15 static double value; /* used to check values */
16 static char num_buf [ 32 ]; /* used to print values */
17
18 static void
19 fill_buf ( sym1, sym2 )
20     symbol
21     sym1,
22     sym2;
23 {
24     /* fills buffer with symbols' text for error reporting */
25     ( void ) strcpy ( buf, systr ( sym1 ) );
26     ( void ) strcat ( buf, " " );
27     ( void ) strcat ( buf, systr ( sym2 ) );
28 } /* fill_buf */
29
30 static void
31 buf_and ( sym1, sym2 )
32     symbol
33     sym1,
34     sym2;
35 {
36     /* fills buffer with symbols' text and "and" for error reporting */
37     ( void ) strcpy ( buf, systr ( sym1 ) );
38     ( void ) strcat ( buf, " and " );
39     ( void ) strcat ( buf, systr ( sym2 ) );
40 } /* buf_and */
41
42 static void
43 do_tech_spec ( T )
44     tree T;
45 {
46     /* processes a technology specification */
47     tree
48     {
49         fkid,
50         lkid,
51         symbol text;
52     }
53     tr_entry ( T );
54     fkid = subtree ( 1, T );
55     lkid = subtree ( 2, T );
56     tr_entry ( fkid );
57     text = textofnode ( fkid );
58     if ( !strcmp ( text, C_Vcc ) )
59     {
60         tr_exit ( fkid );
61         value = check_sum ( lkid );
62         if ( value < 3 || value > 5 )
63             while ( kidcount ( lkid ) )
64             {
65                 lkid = subtree ( 1, lkid );
66                 ( void ) sprintf ( num_buf, "%g", value );
67                 error ( lkid,
68                     /*MSG*/"Vcc out of range: %s (must have 3 <= Vcc <= 5)"
69                     , num_buf );
70             }
71         else if ( the -> Vcc == UNDEFINED )
72             the -> Vcc = value;
73         else
74             error ( fkid,
75                 /*MSG*/"Vcc respecified" );
76     }
77     else if ( !strcmp ( text, C_Vth ) )
78     {
79         tr_exit ( fkid );
80         value = check_sum ( lkid );
81         if ( value < 1.5 || value > 5 )
82             while ( kidcount ( lkid ) )
83             {
84                 lkid = subtree ( 1, lkid );
85                 ( void ) sprintf ( num_buf, "%g", value );
86                 error ( lkid,
87                     /*MSG*/"Vth out of range: %s (must have 1.5 <= Vth <= 5)"
88                     , num_buf );
89             }
90         else if ( the -> Vth == UNDEFINED )
91             the -> Vth = value;
92         else
93             error ( fkid,
94                 /*MSG*/"Vth respecified" );
95     }
96     else if ( !strcmp ( text, C_Vih ) )
97     {
98         tr_exit ( fkid );
99         value = check_sum ( lkid );
100         if ( value < 0 || value > 1 )
101             while ( kidcount ( lkid ) )
102             {
103                 lkid = subtree ( 1, lkid );
104                 ( void ) sprintf ( num_buf, "%g", value );
105                 error ( lkid,
106                     /*MSG*/"Vih out of range: %s (must have 0 <= Vih <= 1)"
107                     , num_buf );
108             }
109         else if ( the -> Vih == UNDEFINED )
110             the -> Vih = value;
111         else
112             error ( fkid,
113                 /*MSG*/"Vih respecified" );
114     }
115     else if ( !strcmp ( text, C_Vih ) )
116     {
117         tr_exit ( fkid );
118         value = check_sum ( lkid );
119         if ( value < 0 || value > 1 )
120             while ( kidcount ( lkid ) )
121             {
122                 lkid = subtree ( 1, lkid );
123                 ( void ) sprintf ( num_buf, "%g", value );
124                 error ( lkid,
125                     /*MSG*/"Vih out of range: %s (must have 0 <= Vih <= 1)"
126                     , num_buf );
127             }
128         else if ( the -> Vih == UNDEFINED )
129             the -> Vih = value;
130         else
131             error ( fkid,
132                 /*MSG*/"Vih respecified" );
133     }
134 }

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_1.c	DATE 5/23/89	PAGE # 2/57
LINE #		SOURCE TEXT	TIME 4:42:25 pm	
121		tr_exit (fkid) ;		
122		value = check_sum (lkid) ;		
123		if (value < 2 value > 5)		
124		{ while (kidcount (lkid)		
125		{ void = subtree (1 , lkid) ;		
126		(void) sprintf (sum_buf , "ig" , value) ;		
127		error		
128		(lkid ,		
129		/*IMSG*/"Vih out of range: ts (must have 2 <= Vih <= 5)"		
130		sum_buf		
131		};		
132		}		
133		else if (the -> Vih == UNDEFINED)		
134		the -> Vih = value ;		
135		else		
136		error		
137		(fkid ,		
138		/*IMSG*/"Vih respecified"		
139		};		
140		}		
141		else if (strcmp (text , C_Vil))		
142		{ set_text (fkid , C_Vil) ;		
143		tr_exit (fkid) ;		
144		value = check_sum (lkid) ;		
145		if (value < 0 value > 2)		
146		{ while (kidcount (lkid)		
147		{ lkid = subtree (1 , lkid) ;		
148		(void) sprintf (sum_buf , "ig" , value) ;		
149		error		
150		(lkid ,		
151		/*IMSG*/"Vil out of range: ts (must have 0 <= Vil <= 2)"		
152		sum_buf		
153		};		
154		}		
155		else if (the -> Vil == UNDEFINED)		
156		the -> Vil = value ;		
157		else		
158		error		
159		(fkid ,		
160		/*IMSG*/"Vil respecified"		
161		};		
162		}		
163		else if (strcmp (text , C_Vah))		
164		{ set_text (fkid , C_Vah) ;		
165		tr_exit (fkid) ;		
166		value = check_sum (lkid) ;		
167		if (value < 1.5 value > 5)		
168		{ while (kidcount (lkid)		
169		{ lkid = subtree (1 , lkid) ;		
170		(void) sprintf (sum_buf , "ig" , value) ;		
171		error		
172		(lkid ,		
173		/*IMSG*/"Vah out of range: ts (must have 1.5 <= Vah <= 5)"		
174		sum_buf		
175		};		
176		}		
177		else if (the -> Vah == UNDEFINED)		
178		the -> Vah = value ;		
179		else		
180		error		
181		(fkid ,		
182		/*IMSG*/"Vah respecified"		
183		};		
184		}		
185		else if (strcmp (text , C_Val))		
186		{ set_text (fkid , C_Val) ;		
187		tr_exit (fkid) ;		
188		value = check_sum (lkid) ;		
189		if (value < 0 value > 1)		
190		{ while (kidcount (lkid)		
191		{ lkid = subtree (1 , lkid) ;		
192		(void) sprintf (sum_buf , "ig" , value) ;		
193		error		
194		(lkid ,		
195		/*IMSG*/"Val out of range: ts (must have 0 <= Val <= 1)"		
196		sum_buf		
197		};		
198		}		
199		else if (the -> Val == UNDEFINED)		
200		the -> Val = value ;		
201		else		
202		error		
203		(fkid ,		
204		/*IMSG*/"Val respecified"		
205		};		
206		}		
207		else if (strcmp (text , C_Ioh))		
208		{ set_text (fkid , C_Ioh) ;		
209		tr_exit (fkid) ;		
210		value = check_sum (lkid) ;		
211		if (value > -0.2)		
212		{ while (kidcount (lkid)		
213		{ lkid = subtree (1 , lkid) ;		
214		(void) sprintf (sum_buf , "ig" , value) ;		
215		error		
216		(lkid ,		
217		/*IMSG*/"Ioh out of range: ts (must have Ioh <= -0.2 mA)"		
218		sum_buf		
219		};		
220		}		
221		else if (the -> Ioh == UNDEFINED)		
222		the -> Ioh = -value ;		
223		else		
224		error		
225		(fkid ,		
226		/*IMSG*/"Ioh respecified"		
227		};		
228		}		
229		else if (strcmp (text , C_Iol))		
230		{ set_text (fkid , C_Iol) ;		
231		tr_exit (fkid) ;		
232		value = check_sum (lkid) ;		
233		if (value < 0.1)		
234		{ while (kidcount (lkid)		
235		{ lkid = subtree (1 , lkid) ;		
236		(void) sprintf (sum_buf , "ig" , value) ;		
237		error		
238		(lkid ,		
239		/*IMSG*/"Iol out of range: ts (must have Iol >= 0.1 mA)"		
240		sum_buf		

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_1.c

DATE

5/23/89

PAGE #

3/58

TIME 4:42:25 pm

```

LINE # SOURCE TEXT
241      }
242      }
243      else if ( the -> Iol == UNDEFINED )
244      the -> Iol = value ;
245      else
246      {
247          error
248          ( fkid ,
249          /*MSG*/"Iol respecified"
250          ) ;
251      }
252      else if ( acompare ( text , C_Iih ) )
253      {
254          set_text ( fkid , C_Iih ) ;
255          tr_exit ( fkid ) ;
256          value = check_num ( lkid ) ;
257          if ( value < 0 || value > 2 )
258          {
259              while ( kidcount ( lkid ) )
260              {
261                  lkid = subtree ( 1 , lkid ) ;
262                  ( void ) sprintf ( sum_buf , "%g" , value ) ;
263                  error
264                  ( lkid ,
265                  /*MSG*/"Iih out of range: %s (must have 0 <= Iih <= 2 mA)"
266                  , sum_buf
267                  ) ;
268              }
269          }
270      }
271      else if ( the -> Iih == UNDEFINED )
272      the -> Iih = value ;
273      else
274      {
275          error
276          ( fkid ,
277          /*MSG*/"Iih respecified"
278          ) ;
279      }
280      else if ( acompare ( text , C_Iil ) )
281      {
282          set_text ( fkid , C_Iil ) ;
283          tr_exit ( fkid ) ;
284          value = check_num ( lkid ) ;
285          if ( value > 0 || value < -2.5 )
286          {
287              while ( kidcount ( lkid ) )
288              {
289                  lkid = subtree ( 1 , lkid ) ;
290                  ( void ) sprintf ( sum_buf , "%g" , value ) ;
291                  error
292                  ( lkid ,
293                  /*MSG*/"Iil out of range: %s (must have 0 >= Iil >= -2.5 mA)"
294                  , sum_buf
295                  ) ;
296              }
297          }
298      }
299      else if ( the -> Iil == UNDEFINED )
300      the -> Iil = -value ;
301      else
302      {
303          error
304          ( fkid ,
305          /*MSG*/"Iil respecified"
306          ) ;
307      }
308      else
309      {
310          error
311          ( fkid ,
312          /*MSG*/"illegal driver specifier: %s (must be Vcc, Iih, or Iil)"
313          , syntostr ( text )
314          ) ;
315          tr_exit ( fkid ) ;
316          ( void ) check_num ( lkid ) ;
317          tr_exit ( T ) ;
318          /* do_tech_spec */
319      }
320
321 static void
322 do_id_attr ( T )
323 {
324     tree T ;
325     /* processes an identifier:attribute */
326     symbol text ;
327     tr_entry ( T ) ;
328     text = textofnode ( T ) ;
329     if ( acompare ( text , C_eval ) )
330     set_text ( T , C_eval ) ;
331     else if ( acompare ( text , C_store ) )
332     set_text ( T , C_store ) ;
333     else if ( acompare ( text , C_edge_rise ) )
334     set_text ( T , C_edge_rise ) ;
335     else if ( acompare ( text , C_edge_fall ) )
336     set_text ( T , C_edge_fall ) ;
337     else if ( acompare ( text , C_feedback ) )
338     set_text ( T , C_feedback ) ;
339     else if ( acompare ( text , C_keepalive ) )
340     set_text ( T , C_keepalive ) ;
341     else if ( acompare ( text , C_pull_up ) )
342     set_text ( T , C_pull_up ) ;
343     else if ( acompare ( text , C_pull_down ) )
344     set_text ( T , C_pull_down ) ;
345     else
346     {
347         error
348         ( T ,
349         /*MSG*/"illegal pin attribute: %s (must be eval, store, edge_rise, edge_fall, feedback, keepalive, pull_up, or pull_down)"
350         , syntostr ( text )
351         ) ;
352         tr_exit ( T ) ;
353     }
354     /* do_id_attr */
355 }
356
357 static void
358 do_resistance ( T )
359 {
360     tree T ;
361     unsigned i , len , kohms = 0 ;
362     double ohms ;
363     symbol old ;
364     ( void ) strcpy ( buf , syntostr ( old = textofnode ( T ) ) ) ;
365     len = strlen ( buf ) - 1 ;
366     if ( buf [ len ] == 'k' || buf [ len ] == 'K' )
367     {
368         for ( i = 0 ; i < len ; ++i )
369         {
370             if ( buf [ i ] < '0' || buf [ i ] > '9' )
371             {
372                 buf [ i ] = '.' ;
373                 if ( buf [ i ] != '.' )
374                     break ;
375             }
376             if ( i == len )
377             {
378                 buf [ len ] = '\0' ;
379                 set_text ( T , strtosym ( buf ) ) ;
380                 kohms = 1 ;
381             }
382         }
383     }
384     ohms = check_num ( T ) ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE 5/23/89 PAGE #
TIME 4:42:25 pm 4/59

```

LINE # SOURCE TEXT
361 if ( kobs )
362 { obs = 1000 ;
363   set_text ( T , old ) ;
364 }
365 decorate ( T , ( long ) ( obs + 0.5 ) ) ;
366 /* do_resistance */
367
368 static void
369 do_current_spec ( T )
370 {
371   tree T ;
372   /* processes a current or resistance specification */
373   symbol text ;
374   tr_entry ( T ) ;
375   kid = subtree ( 1 , T ) ;
376   tr_entry ( kid ) ;
377   text = textofnode ( kid ) ;
378   if ( strcmp ( text , C_Ioh ) )
379   { set_text ( kid , C_Ioh ) ;
380     tr_exit ( kid ) ;
381     kid = subtree ( 2 , T ) ;
382     value = check_sum ( kid ) ;
383     if ( value > -0.2 )
384     { while ( kidcount ( kid ) )
385       { kid = subtree ( 1 , kid ) ;
386         ( void ) sprintf ( num_buf , "%g" , value ) ;
387         perror
388           ( kid ,
389             /*MSG*/"Ioh out of range: %s (must have Ioh <= -0.2 mA)"
390             , num_buf
391           ) ;
392       }
393     }
394   }
395   else if ( strcmp ( text , C_Iol ) )
396   { set_text ( kid , C_Iol ) ;
397     tr_exit ( kid ) ;
398     kid = subtree ( 2 , T ) ;
399     value = check_sum ( kid ) ;
400     if ( value < 0.1 )
401     { while ( kidcount ( kid ) )
402       { kid = subtree ( 1 , kid ) ;
403         ( void ) sprintf ( num_buf , "%g" , value ) ;
404         perror
405           ( kid ,
406             /*MSG*/"Iol out of range: %s (must have Iol >= 0.1 mA)"
407             , num_buf
408           ) ;
409       }
410     }
411   }
412   else if ( strcmp ( text , C_Iih ) )
413   { set_text ( kid , C_Iih ) ;
414     tr_exit ( kid ) ;
415     kid = subtree ( 2 , T ) ;
416     value = check_sum ( kid ) ;
417     if ( value < 0 || value > 2 )
418     { while ( kidcount ( kid ) )
419       { kid = subtree ( 1 , kid ) ;
420         ( void ) sprintf ( num_buf , "%g" , value ) ;
421         perror
422           ( kid ,
423             /*MSG*/"Iih out of range: %s (must have 0 <= Iih <= 2 mA)"
424             , num_buf
425           ) ;
426       }
427     }
428   }
429   else if ( strcmp ( text , C_Iil ) )
430   { set_text ( kid , C_Iil ) ;
431     tr_exit ( kid ) ;
432     kid = subtree ( 2 , T ) ;
433     value = check_sum ( kid ) ;
434     if ( value > 0 || value < -2.5 )
435     { while ( kidcount ( kid ) )
436       { kid = subtree ( 1 , kid ) ;
437         ( void ) sprintf ( num_buf , "%g" , value ) ;
438         perror
439           ( kid ,
440             /*MSG*/"Iil out of range: %s (must have 0 >= Iil >= -2.5 mA)"
441             , num_buf
442           ) ;
443       }
444     }
445   }
446   else if ( strcmp ( text , C_pull_up ) )
447   { set_text ( kid , C_pull_up ) ;
448     tr_exit ( kid ) ;
449     kid = subtree ( 2 , T ) ;
450     do_resistance ( kid ) ;
451   }
452   else if ( strcmp ( text , C_pull_down ) )
453   { set_text ( kid , C_pull_down ) ;
454     tr_exit ( kid ) ;
455     kid = subtree ( 2 , T ) ;
456     do_resistance ( kid ) ;
457   }
458   else
459   { perror
460     ( kid ,
461       /*MSG*/"illegal pin attribute: %s (must be Iih, Iil, pull_up, or pull_down)"
462       , syntostr ( text )
463     ) ;
464     tr_exit ( kid ) ;
465     kid = subtree ( 2 , T ) ;
466     ( void ) check_sum ( kid ) ;
467   }
468   tr_exit ( T ) ;
469 } /* do_current_spec */
470
471 static void
472 do_cycle_spec ( T )
473 {
474   tree T ;
475   /* processes a cycle specification */
476   { tree kid ;
477     symbol text ;
478     tr_entry ( T ) ;
479     kid = subtree ( 1 , T ) ;
480     tr_entry ( kid ) ;
481     text = textofnode ( kid ) ;
482     if ( strcmp ( text , C_in_sequence ) )
483     { set_text ( kid , C_in_sequence ) ;
484     }
485     else if ( strcmp ( text , C_last_cycle ) )
486     { set_text ( kid , C_last_cycle ) ;
487     }
488   }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_1.c

DATE 5/23/89

PAGE #

TIME 4:42:25 pm

5/60

```

LINE # SOURCE TEXT
481     else
482     {
483     { kid ,
484     /*MSG*/"illegal cycle specifier: ts (must be in_sequence or last_cycle)"
485     , syntostr ( text )
486     }
487     }
488     tr_exit ( kid ) ;
489     kid = subtree ( 2 , T ) ;
490     tr_entry ( kid ) ;
491     text = textofnode ( kid ) ;
492     if ( ! compare ( text , C_hard ) )
493     {
494     set_text ( kid , C_hard ) ;
495     }
496     else if ( ! compare ( text , C_medium ) )
497     {
498     set_text ( kid , C_medium ) ;
499     }
500     else if ( ! compare ( text , C_hard_medium ) )
501     {
502     set_text ( kid , C_hard_medium ) ;
503     }
504     else
505     {
506     error
507     {
508     { kid ,
509     /*MSG*/"illegal cycle specification: ts (must be hard, medium, or hard_medium)"
510     , syntostr ( text )
511     }
512     }
513     tr_exit ( kid ) ;
514     tr_exit ( T ) ;
515     } /* do_cycle_spec */
516
517 static void
518 do_in_pins ( T )
519 {
520     tree T ;
521     /* counts and checks attributes of input pins */
522     { ushort
523     {
524     {
525     kids ,
526     k ,
527     count = 0 ;
528     tree kid ,
529     symbol text ,
530     tr_entry ( T ) ,
531     kids = kidcount ( T ) ,
532     for
533     {
534     { kid = subtree ( 1 = 1 , T ) ;
535     nameofnode ( kid ) != N_NAMES ;
536     kid = subtree ( ++i , T ) ;
537     }
538     count ++ do_pin_name ( kid ) ;
539     the -> sum_in ++ count ;
540     tr_entry ( kid ) ;
541     k = kidcount ( kid ) ;
542     if ( k != count )
543     {
544     buf_instant ( k , count ) ;
545     error
546     {
547     { subtree ( k , kid ) ,
548     /*MSG*/"wrong number of in_pin numbers compared to device signal names: ts"
549     , buf
550     }
551     }
552     tr_exit ( kid ) ;
553     while ( ++i <= kids )
554     {
555     kid = subtree ( i , T ) ;
556     switch ( nameofnode ( kid ) )
557     {
558     case P_ID :
559     {
560     do_id_attr ( kid ) ;
561     text = textofnode ( kid ) ;
562     if ( text == C_feedback )
563     {
564     error
565     {
566     { kid ,
567     /*MSG*/"ts attribute not allowed on in_pins"
568     , syntostr ( text )
569     }
570     }
571     }
572     else if ( text == C_pull_up || text == C_pull_down )
573     {
574     error
575     {
576     { kid ,
577     /*MSG*/"ts attribute on an in_pin: must declare as an io_pin"
578     , syntostr ( text )
579     }
580     }
581     }
582     else if
583     {
584     { text == C_eval ||
585     text == C_store ||
586     text == C_edge_rise ||
587     text == C_edge_fall
588     }
589     the -> has_store_pins = 1 ;
590     break ;
591     case N_ATTR :
592     {
593     if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID )
594     {
595     do_cycle_spec ( kid ) ;
596     error
597     {
598     { subtree ( 1 , kid ) ,
599     /*MSG*/"cycle specification attribute illegal on in_pins"
600     , syntostr ( text )
601     }
602     }
603     }
604     }
605     }
606     }
607     }
608     }
609     }
610     }
611     }
612     }
613     }
614     }
615     }
616     }
617     }
618     }
619     }
620     }
621     }
622     }
623     }
624     }
625     }
626     }
627     }
628     }
629     }
630     }
631     }
632     }
633     }
634     }
635     }
636     }
637     }
638     }
639     }
640     }
641     }
642     }
643     }
644     }
645     }
646     }
647     }
648     }
649     }
650     }
651     }
652     }
653     }
654     }
655     }
656     }
657     }
658     }
659     }
660     }
661     }
662     }
663     }
664     }
665     }
666     }
667     }
668     }
669     }
670     }
671     }
672     }
673     }
674     }
675     }
676     }
677     }
678     }
679     }
680     }
681     }
682     }
683     }
684     }
685     }
686     }
687     }
688     }
689     }
690     }
691     }
692     }
693     }
694     }
695     }
696     }
697     }
698     }
699     }
700     }
701     }
702     }
703     }
704     }
705     }
706     }
707     }
708     }
709     }
710     }
711     }
712     }
713     }
714     }
715     }
716     }
717     }
718     }
719     }
720     }
721     }
722     }
723     }
724     }
725     }
726     }
727     }
728     }
729     }
730     }
731     }
732     }
733     }
734     }
735     }
736     }
737     }
738     }
739     }
740     }
741     }
742     }
743     }
744     }
745     }
746     }
747     }
748     }
749     }
750     }
751     }
752     }
753     }
754     }
755     }
756     }
757     }
758     }
759     }
760     }
761     }
762     }
763     }
764     }
765     }
766     }
767     }
768     }
769     }
770     }
771     }
772     }
773     }
774     }
775     }
776     }
777     }
778     }
779     }
780     }
781     }
782     }
783     }
784     }
785     }
786     }
787     }
788     }
789     }
790     }
791     }
792     }
793     }
794     }
795     }
796     }
797     }
798     }
799     }
800     }
801     }
802     }
803     }
804     }
805     }
806     }
807     }
808     }
809     }
810     }
811     }
812     }
813     }
814     }
815     }
816     }
817     }
818     }
819     }
820     }
821     }
822     }
823     }
824     }
825     }
826     }
827     }
828     }
829     }
830     }
831     }
832     }
833     }
834     }
835     }
836     }
837     }
838     }
839     }
840     }
841     }
842     }
843     }
844     }
845     }
846     }
847     }
848     }
849     }
850     }
851     }
852     }
853     }
854     }
855     }
856     }
857     }
858     }
859     }
860     }
861     }
862     }
863     }
864     }
865     }
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }
946     }
947     }
948     }
949     }
950     }
951     }
952     }
953     }
954     }
955     }
956     }
957     }
958     }
959     }
960     }
961     }
962     }
963     }
964     }
965     }
966     }
967     }
968     }
969     }
970     }
971     }
972     }
973     }
974     }
975     }
976     }
977     }
978     }
979     }
980     }
981     }
982     }
983     }
984     }
985     }
986     }
987     }
988     }
989     }
990     }
991     }
992     }
993     }
994     }
995     }
996     }
997     }
998     }
999     }
1000    }

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE	5/23/89	PAGE #
TIME	4:42:25 pm	6/61

```

LINE #          SOURCE TEXT
601  tree T ;
602  /* counts and checks attributes of output pins */
603  {
604      ushort
605      kids ,
606      k ,
607      count = 0 ;
608      tree kid ;
609      symbol text ;
610      tr_entry ( T ) ;
611      kids = kidcount ( T ) ;
612      for
613      ( kid = subtree ( 1 = 1 , T ) ;
614        nameofnode ( kid ) != M_NAMES ;
615        kid = subtree ( ++1 , T )
616      )
617      count ++ do_pin_name ( kid ) ;
618      the -> num_out ++ count ;
619      tr_entry ( kid ) ;
620      k = kidcount ( kid ) ;
621      if ( k != count )
622      {
623          buf_instant ( k , count ) ;
624          error
625          ( subtree ( k , kid ) ,
626            /*MSG*/"wrong number of out_pin numbers compared to device signal names: is"
627            , buf
628          ) ;
629      }
630      while ( ++i <= kids )
631      {
632          kid = subtree ( i , T ) ;
633          switch ( nameofnode ( kid ) )
634          {
635              case P_ID :
636                  do_id_attr ( kid ) ;
637                  text = textofnode ( kid ) ;
638                  if
639                  (
640                      text == C_eval ||
641                      text == C_store ||
642                      text == C_edge_rise ||
643                      text == C_edge_fall ||
644                      text == C_keepalive
645                  )
646                  error
647                  ( kid ,
648                    /*MSG*/"is attribute not allowed on out_pins"
649                    , syntostr ( text )
650                  ) ;
651                  if ( text == C_feedback )
652                      the -> has_feedback_pins = 1 ;
653                  break ;
654              case M_ATTR :
655                  if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID )
656                      ( do_cycle_spec ( kid ) ;
657                      error
658                      ( subtree ( 1 , kid ) ,
659                        /*MSG*/"cycle specification attribute illegal on out_pins"
660                      ) ;
661                      break ;
662                      do_current_spec ( kid ) ;
663                      text = textofnode ( subtree ( 1 , kid ) ) ;
664                      if ( text == C_iib || text == C_iil )
665                      {
666                          error
667                          ( subtree ( 1 , kid ) ,
668                            /*MSG*/"is current specifier not allowed on out_pins"
669                          ) ;
670                          syntostr ( text )
671                          ,
672                          break ;
673                          default :
674                          error
675                          ( kid ,
676                            /*MSG*/"internal error: do_out_pins: unknown node name: is"
677                          ) ;
678                          syntostr ( nameofnode ( kid ) )
679                          ,
680                          break ;
681                      }
682                  }
683          }
684      }
685      tr_exit ( T ) ;
686      } /* do_out_pins */
687
688  static void
689  do_io_pins ( T )
690  {
691      tree T ;
692      /* counts and checks attributes of I/O pins */
693      {
694          ushort
695          kids ,
696          k ,
697          count = 0 ;
698          tree kid ;
699          symbol text ;
700          tr_entry ( T ) ;
701          kids = kidcount ( T ) ;
702          for
703          ( kid = subtree ( 1 = 1 , T ) ;
704            nameofnode ( kid ) != M_NAMES ;
705            kid = subtree ( ++1 , T )
706          )
707          count ++ do_pin_name ( kid ) ;
708          the -> num_io ++ count ;
709          tr_entry ( kid ) ;
710          k = kidcount ( kid ) ;
711          if ( k != count )
712          {
713              buf_instant ( k , count ) ;
714              error
715              ( subtree ( k , kid ) ,
716                /*MSG*/"wrong number of io_pin numbers compared to device signal names: is"
717                , buf
718              ) ;
719          }
720          while ( ++i <= kids )
721          {
722              kid = subtree ( i , T ) ;
723              switch ( nameofnode ( kid ) )
724              {
725                  case P_ID :
726                      do_id_attr ( kid ) ;
727                      text = textofnode ( kid ) ;
728                      if ( text == C_feedback )
729                          the -> has_feedback_pins = 1 ;
730                      if

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE 5/23/89
TIME 4:42:25 pm

PAGE #
7/62

```

LINE #          SOURCE TEXT
721          ( text == C_eval ||
722            text == C_star ||
723            text == C_edge_rise ||
724            text == C_edge_fall
725          )
726          the -> has_store_pins = 1 ;
727          break ;
728          case M_ATTR :
729            if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID_ )
730              do_cycle_spec ( kid ) ;
731            else
732              do_current_spec ( kid ) ;
733            break ;
734          default :
735            error
736              ( kid ,
737              /*MSG*/"internal error: do_io_pins: unknown node name: %s"
738              , syntestr ( nameofnode ( kid ) )
739              ) ;
740            break ;
741          }
742          tr_exit ( T ) ;
743          } /* do_io_pins */
744
745          static void
746          do_power_pins ( T )
747          {
748            tree T ;
749            /* counts and checks attributes of power pins */
750            { ushort
751              i ,
752              kids ,
753              k ,
754              count = 0 ,
755              tree kid ,
756              tr_entry ( T ) ,
757              kids = kidcount ( T ) ;
758              for
759                ( kid = subtree ( i = 1 , T ) ;
760                  nameofnode ( kid ) != M_NAMES ;
761                  kid = subtree ( ++i , T )
762                )
763                  count += do_pin_name ( kid ) ;
764              tr_entry ( kid ) ;
765              k = kidcount ( kid ) ;
766              if ( count == 1 && k != 1 )
767                { decorate ( subtree ( 1 , T ) , ( long ) k ) ;
768                  the -> sum_power += k ;
769                }
770              else
771                the -> sum_power += count ;
772              if ( count != 1 && k != count )
773                { buf_instead ( k , count ) ;
774                  error
775                    ( subtree ( k , kid ) ,
776                    /*MSG*/"wrong number of power_pin numbers compared to device signal names: %s"
777                    , buf
778                    ) ;
779                }
780              tr_exit ( kid ) ;
781              while ( ++i <= kids )
782                { kid = subtree ( i , T ) ;
783                  switch ( nameofnode ( kid ) )
784                    { case P_ID_ :
785                      do_id_attr ( kid ) ;
786                      error
787                        ( kid ,
788                        /*MSG*/"power_pin attributes illegal: %s"
789                        , sof ( kid )
790                        ) ;
791                      break ;
792                    case M_ATTR :
793                      if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID_ )
794                        { do_cycle_spec ( kid ) ;
795                          error
796                            ( subtree ( 1 , kid ) ,
797                            /*MSG*/"cycle specification attribute illegal on power_pins"
798                            ) ;
799                          break ;
800                        }
801                      do_current_spec ( kid ) ;
802                      error
803                        ( subtree ( 1 , kid ) ,
804                        /*MSG*/"current or resistance specification attribute illegal on power_pins"
805                        ) ;
806                      break ;
807                    default :
808                      error
809                        ( kid ,
810                        /*MSG*/"internal error: do_power_pins: unknown node name: %s"
811                        , syntestr ( nameofnode ( kid ) )
812                        ) ;
813                      break ;
814                    }
815                }
816              tr_exit ( T ) ;
817            } /* do_power_pins */
818
819          static
820          do_ground_pins ( T )
821          {
822            tree T ;
823            /* counts and checks attributes of ground pins */
824            { ushort
825              i ,
826              kids ,
827              k ,
828              count = 0 ,
829              tree kid ,
830              tr_entry ( T ) ,
831              kids = kidcount ( T ) ;
832              for
833                ( kid = subtree ( i = 1 , T ) ;
834                  nameofnode ( kid ) != M_NAMES ;
835                  kid = subtree ( ++i , T )
836                )
837                  count += do_pin_name ( kid ) ;
838              tr_entry ( kid ) ;
839              k = kidcount ( kid ) ;
840              if ( count == 1 && k != 1 )
841                { decorate ( subtree ( 1 , T ) , ( long ) k ) ;

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE 5/23/89
TIME 4:42:25 pm

PAGE #
8/63

```

LINE # SOURCE TEXT
841 the -> sum_ground += k ;
842 }
843 else
844 the -> sum_ground += count ;
845 if ( count != 1 && k != count )
846 { buf_instead ( k , count ) ;
847 error
848 ( subtree ( k , kid ) ,
849 /*MSG*/"wrong number of ground_pin numbers compared to device signal names: ts"
850 , buf
851 ) ;
852 }
853 tr_exit ( kid ) ;
854 while ( ++i <= kids )
855 { kid = subtree ( i , T ) ;
856 switch ( nameofnode ( kid ) )
857 { case P_ID_ :
858 do_id_attr ( kid ) ;
859 error
860 ( kid ,
861 /*MSG*/"ground_pin attributes illegal: ts"
862 , sof ( kid )
863 ) ;
864 break ;
865 case M_ATTR :
866 if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID_ )
867 { do_cycle_spec ( kid ) ;
868 error
869 ( subtree ( 1 , kid ) ,
870 /*MSG*/"cycle specification attribute illegal on ground_pins"
871 ) ;
872 break ;
873 }
874 do_current_spec ( kid ) ;
875 error
876 ( subtree ( 1 , kid ) ,
877 /*MSG*/"current or resistance specification attribute illegal on ground_pins"
878 ) ;
879 break ;
880 default :
881 error
882 ( kid ,
883 /*MSG*/"internal error: do_ground_pins: unknown node name: ts"
884 , syntostr ( nameofnode ( kid ) )
885 ) ;
886 break ;
887 }
888 tr_exit ( T ) ;
889 } /*do_ground_pins*/
890
891 static void
892 do_nc_pins ( T )
893 {
894 tree T ;
895 /* counts and checks attributes of NC pins */
896 { ushort
897 i
898 kids ,
899 k ,
900 count = 0 ;
901 tree kid ;
902 tr_entry ( T ) ;
903 kids = kidcount ( T ) ;
904 for
905 { kid = subtree ( i = 1 , T ) ;
906 nameofnode ( kid ) != N_NAMES ;
907 kid = subtree ( ++i , T )
908 }
909 count += do_pin_name ( kid ) ;
910 tr_entry ( kid ) ;
911 k = kidcount ( kid ) ;
912 if ( count == 1 && k != 1 )
913 { decorate ( subtree ( 1 , T ) , ( long ) k ) ;
914 the -> sum_nc += k ;
915 }
916 else
917 the -> sum_nc += count ;
918 if ( count != 1 && k != count )
919 { buf_instead ( k , count ) ;
920 error
921 ( subtree ( k , kid ) ,
922 /*MSG*/"wrong number of nc_pin numbers compared to device signal names: ts"
923 , buf
924 ) ;
925 }
926 tr_exit ( kid ) ;
927 while ( ++i <= kids )
928 { kid = subtree ( i , T ) ;
929 switch ( nameofnode ( kid ) )
930 { case P_ID_ :
931 do_id_attr ( kid ) ;
932 error
933 ( kid ,
934 /*MSG*/"nc_pin attributes illegal: ts"
935 , sof ( kid )
936 ) ;
937 break ;
938 case M_ATTR :
939 if ( nameofnode ( subtree ( 2 , kid ) ) == P_ID_ )
940 { do_cycle_spec ( kid ) ;
941 error
942 ( subtree ( 1 , kid ) ,
943 /*MSG*/"cycle specification attribute illegal on nc_pins"
944 ) ;
945 break ;
946 }
947 do_current_spec ( kid ) ;
948 error
949 ( subtree ( 1 , kid ) ,
950 /*MSG*/"current or resistance specification attribute illegal on nc_pins"
951 ) ;
952 break ;
953 default :
954 error
955 ( kid ,
956 /*MSG*/"internal error: do_nc_pins: unknown node name: ts"
957 , syntostr ( nameofnode ( kid ) )
958 ) ;
959 break ;
960 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_1.c

DATE

5/23/89

PAGE #

TIME 4:42:25 pm

9/64

```

LINE # SOURCE TEXT
961 }
962 tr_exit ( T ) ;
963 } /* do_bc_pina */
964
965 static ushort
966 do_resets ( T )
967 tree T ;
968 /* process reset pins, return how many */
969 { ushort
970 k ;
971 count = 0 ;
972 kids ;
973 tr_entry ( T ) ;
974 kids = kidcount ( T ) ;
975 for ( k = 1 ; k < kids ; ++k )
976 count += do_pina_sub ( subtree ( k , T ) ) ;
977 tr_exit ( T ) ;
978 return count ;
979 } /* do_resets */
980
981 static unsigned long
982 do_unit ( T , unit )
983 tree T ;
984 symbol *unit ;
985 /* check clock rate units, return time in picoseconds, and unit identifier */
986 { tree kid ;
987 char *str , *s ;
988 symbol text ;
989 double val ;
990 ushort set = 0 ;
991 *unit = NULLSYM ;
992 tr_entry ( T ) ;
993 kid = subtree ( 1 , T ) ;
994 if ( kidcount ( T ) == 1 )
995 { text = textofnode ( kid ) ;
996 str = symtostr ( text ) ;
997 s = buf ;
998 while ( *str < 'A' || *str > 'Z' ) && ( *str < 'a' || *str > 'z' )
999 *s++ = *str++ ;
1000 *s = '\0' ;
1001 set_text ( kid , strtosym ( buf ) ) ;
1002 val = check_sum ( kid ) ;
1003 set_text ( kid , text ) ;
1004 text = strtosym ( str ) ;
1005 }
1006 else
1007 { val = check_sum ( kid ) ;
1008 kid = subtree ( 2 , T ) ;
1009 tr_entry ( kid ) ;
1010 text = textofnode ( kid ) ;
1011 set = 1 ;
1012 tr_exit ( kid ) ;
1013 }
1014 if ( strcmp ( text , C_Hz ) )
1015 { *unit = C_Hz ;
1016 if ( set )
1017 set_text ( kid , C_Hz ) ;
1018 if ( val != 0 )
1019 val = 1 / val * 1E12 ;
1020 }
1021 else if ( strcmp ( text , C_MHz ) )
1022 { *unit = C_MHz ;
1023 if ( set )
1024 set_text ( kid , C_MHz ) ;
1025 if ( val != 0 )
1026 val = 1 / val * 1E9 ;
1027 }
1028 else if ( strcmp ( text , C_kHz ) )
1029 { *unit = C_kHz ;
1030 if ( set )
1031 set_text ( kid , C_kHz ) ;
1032 if ( val != 0 )
1033 val = 1 / val * 1E6 ;
1034 }
1035 else if ( strcmp ( text , C_GHz ) )
1036 { *unit = C_GHz ;
1037 if ( set )
1038 set_text ( kid , C_GHz ) ;
1039 if ( val != 0 )
1040 val = 1 / val * 1E3 ;
1041 }
1042 else if ( strcmp ( text , C_ms ) )
1043 { *unit = C_ms ;
1044 if ( set )
1045 set_text ( kid , C_ms ) ;
1046 if ( val != 0 )
1047 val = 1E9 / val ;
1048 }
1049 else
1050 error
1051 ( kid ,
1052 /*IMSC*/"zero clock period illegal"
1053 ) ;
1054 }
1055 else if ( strcmp ( text , C_us ) )
1056 { *unit = C_us ;
1057 if ( set )
1058 set_text ( kid , C_us ) ;
1059 if ( val != 0 )
1060 val = 1E6 / val ;
1061 }
1062 else
1063 error
1064 ( kid ,
1065 /*IMSC*/"zero clock period illegal"
1066 ) ;
1067 }
1068 else if ( strcmp ( text , C_ns ) )
1069 { *unit = C_ns ;
1070 if ( set )
1071 set_text ( kid , C_ns ) ;
1072 if ( val != 0 )
1073 val = 1E3 / val ;
1074 }
1075 else
1076 error
1077 ( kid ,
1078 /*IMSC*/"zero clock period illegal"
1079 ) ;
1080 }
1081 else if ( strcmp ( text , C_ps ) )
1082 { *unit = C_ps ;
1083 if ( set )

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_1.c	DATE 5/23/89 TIME 4:42:25 pm	PAGE # 10/65
LINE #	SOURCE TEXT			
1081	set_text (kid , C_ps) ;			
1082	if (val == 0)			
1083	error			
1084	(kid ,			
1085	/*LMSC*/"zero clock period illegal"			
1086)			
1087	}			
1088	else			
1089	{ error			
1090	(kid ,			
1091	/*LMSC*/"illegal unit specifier: %s (must be KHz, MHz, uS, or nS)"			
1092	, syntostr (text)			
1093)			
1094	val = 0 ;			
1095	}			
1096	if (val != 0)			
1097	if (val > ABS_MAX_PERIOD)			
1098	val = 0 ;			
1099	else if (val < ABS_MIN_PERIOD)			
1100	{ fill_buf (textofnode (subtree (1 , T)) , textofnode (kid)) ;			
1101	{ void) sprintf			
1102	(num_buf ,			
1103	"alu",			
1104	(unsigned long) ABS_MIN_PERIOD / 1000			
1105)			
1106	error			
1107	(kid ,			
1108	/*LMSC*/"device speed too high: %s (period must be no less than %s)"			
1109	, buf ,			
1110	num_buf			
1111)			
1112	val = 0 ;			
1113	}			
1114	tr_exit (T) ;			
1115	return val + 0.999 ;			
1116	} /* do_unit */			
1117	}			
1118	usboot			
1119	check_pin_number (num)			
1120	tree num ;			
1121	/* checks and returns the passed device adapter pin number node */			
1122	{ char			
1123	s ;			
1124	-str ;			
1125	unsigned long val = 0 ;			
1126	s =			
1127	str =			
1128	syntostr (textofnode (num)) ;			
1129	if ('s' >= '1' && 's' <= '9')			
1130	{ val = 's' - '0' ;			
1131	while (++s >= '0' && 's' <= '9')			
1132	if (val < MAX_PINS)			
1133	if (val == 10)			
1134	val += 's' - '0' ;			
1135	}			
1136	if ('s' != '\0')			
1137	{ val = 0 ;			
1138	error			
1139	(num ,			
1140	/*LMSC*/"illegal device adapter pin number (non-numeric characters): %s"			
1141	, str			
1142)			
1143	}			
1144	}			
1145	else if ('s' != '0' s[1] != '\0')			
1146	error			
1147	(num ,			
1148	/*LMSC*/"illegal device adapter pin number (leading zeros): %s"			
1149	, str			
1150)			
1151	if (val >= MAX_PINS)			
1152	{ val = 0 ;			
1153	{ void) sprintf (num_buf , "lu", (unsigned long) MAX_PINS) ;			
1154	error			
1155	(num ,			
1156	/*LMSC*/"device adapter pin number too large: %s (must be less than %s)"			
1157	, str			
1158)			
1159	}			
1160	return val ;			
1161	} /* check_pin_number */			
1162	static void			
1163	amapit (from , to , parent)			
1164	tree			
1165	from			
1166	to ,			
1167	parent ;			
1168	/* process mapping for a single device adapter mapping */			
1169	{ tree look ,			
1170	usboot num ;			
1171	symbol			
1172	text1 ,			
1173	text2 ,			
1174	tr_entry (from) ;			
1175	text2 = textofnode (from) ;			
1176	text1 = textofnode (from) ;			
1177	if			
1178	{ (look = g_lookup (text1)) != NULLTREE &&			
1179	nameofnode (look) == M_ADAPTER			
1180	{			
1181	error			
1182	(from ,			
1183	/*LMSC*/"device adapter mapping respecified for device adapter pin name %s"			
1184	, syntostr (text1)			
1185)			
1186	else			
1187	g_declare (text1 , parent) ;			
1188	tr_exit (from) ;			
1189	tr_entry (to) ;			
1190	text2 = textofnode (to) ;			
1191	if ((num = check_pin_number (to)) >= the -> spin_cnt)			
1192	the -> spin_cnt = num + 1 ;			
1193	while (spin_number_of (text2))			
1194	text2 = prepend_underscore (text2 , 0) ;			
1195	set_spin_number (text2 , text1) ;			
1196	tr_exit (to) ;			
1197	} /* amapit */			
1198	static void			
1199	do_mapping (T , parent)			
1200	{			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_1.c

DATE

5/23/89

PAGE #

TIME

4:42:25 pm

11/66

```

1201  tree
1202  T ,
1203  parent ,
1204  /* process device adapter mappings */
1205  { tree
1206  kid1 ,
1207  kid2 ,
1208  ushort
1209  k ,
1210  kidal ,
1211  kids2 ,
1212  tr_entry ( T ) ,
1213  kid1 = subtree ( 1 , T ) ,
1214  tr_entry ( kid1 ) ,
1215  kidal = kidcount ( kid1 ) ,
1216  kid2 = subtree ( 2 , T ) ,
1217  kids2 = kidcount ( kid2 ) ,
1218  if ( kidal > kids2 )
1219  { buf_instead ( kids2 , kidal ) ,
1220  error
1221  }
1222  /*IMSG*/"too few device adapter pin numbers: %s"
1223  , buf
1224  } ,
1225  }
1226  else if ( kidal < kids2 )
1227  error
1228  { subtree ( kidal + 1 , kid2 ) ,
1229  /*IMSG*/"too many device adapter pin numbers: %s"
1230  , syntostr ( textofnode ( subtree ( kidal + 1 , kids2 ) ) )
1231  } ,
1232  tr_exit ( kid1 ) ,
1233  tr_entry ( kid2 ) ,
1234  for ( k = 1 ; k <= kidal ; ++k )
1235  if ( k <= kids2 )
1236  mapit ( subtree ( k , kid1 ) , subtree ( k , kid2 ) , parent ) ,
1237  tr_exit ( kid2 ) ,
1238  tr_exit ( T ) ,
1239  } /* do_mapping */
1240
1241  void
1242  walk_1 ( T )
1243  tree T ,
1244  /* the constraints first walk */
1245  { ushort
1246  k ,
1247  kids ,
1248  unsigned long
1249  ratel ,
1250  rate2 ,
1251  tw ,
1252  tree
1253  fkid ,
1254  lkid ,
1255  symbol
1256  name ,
1257  text ,
1258  tr_entry ( T ) ,
1259  name = nameofnode ( T ) ,
1260  kids = kidcount ( T ) ,
1261  if ( kids )
1262  fkid = subtree ( 1 , T ) ,
1263  if ( kids > 1 )
1264  lkid = subtree ( kids , T ) ,
1265  switch ( name )
1266  {
1267  /* .MDL statements */
1268  case M_NAME :
1269  tr_entry ( fkid ) ,
1270  if ( the -> dev_name == NULLSYM )
1271  if
1272  { *syntostr ( textofnode ( fkid ) ) != "" &&
1273  *syntostr ( textofnode ( lkid ) ) != ""
1274  }
1275  the -> dev_name = textofnode ( fkid ) ,
1276  else
1277  { buf_string ( syntostr ( textofnode ( fkid ) ) ) ,
1278  the -> dev_name = strtosys ( buf ) ,
1279  }
1280  else
1281  error
1282  { fkid ,
1283  /*IMSG*/"internal error: device_name respesified: %s"
1284  , sof ( fkid )
1285  } ,
1286  tr_exit ( fkid ) ,
1287  break ,
1288  /* .OPT statements */
1289  case M_FAST :
1290  if ( the -> ultra_fast )
1291  error
1292  { T
1293  /*IMSG*/"ultra_fast respesified"
1294  } ,
1295  else
1296  the -> ultra_fast = 1 ,
1297  break ,
1298  case M_USAGE :
1299  tr_entry ( fkid ) ,
1300  text = textofnode ( fkid ) ,
1301  if ( !strcmp ( text , C_public ) )
1302  set_text ( fkid , C_public ) ,
1303  else if ( !strcmp ( text , C_private ) )
1304  set_text ( fkid , C_private ) ,
1305  else
1306  error
1307  { fkid ,
1308  /*IMSG*/"illegal device_usage: %s (must be public or private)"
1309  , syntostr ( text )
1310  } ,
1311  if ( the -> dev_type == NULLSYM )
1312  the -> dev_type = textofnode ( fkid ) ,
1313  else
1314  error
1315  { fkid ,
1316  /*IMSG*/"internal error: device_usage respesified: %s"
1317  , syntostr ( text )
1318  } ,
1319  tr_exit ( fkid ) ,
1320  break ,

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_1.c	DATE 5/23/89	PAGE # 12/67
LINE #	SOURCE TEXT			
1321	case M_REPORT :			
1322	tr_entry (fkid) ,			
1323	text = textofnode (fkid) ,			
1324	if (! compare (text , C_missing_delays))			
1325	set_text (fkid , C_missing_delays) ;			
1326	else if (! compare (text , C_io_store_changes))			
1327	set_text (fkid , C_io_store_changes) ;			
1328	else			
1329	error			
1330	(fkid ,			
1331	/*LMSC*/"illegal report specifier: %s (must be missing_delays or io_store_changes)"			
1332	, syntostr (text)			
1333) ,			
1334	name = textofnode (fkid) ,			
1335	tr_exit (fkid) ,			
1336	if (klds == 1)			
1337	{ tr_entry (lkid) ;			
1338	text = textofnode (lkid) ,			
1339	if (! compare (text , C_on))			
1340	set_text (lkid , C_on) ;			
1341	else if (! compare (text , C_off))			
1342	set_text (lkid , C_off) ;			
1343	else			
1344	{ error			
1345	(lkid ,			
1346	/*LMSC*/"illegal report command: %s (must be on or off)"			
1347	, syntostr (text)			
1348) ,			
1349	text = NULLSYM ;			
1350	}			
1351	text = textofnode (lkid) ,			
1352	tr_exit (lkid) ,			
1353	}			
1354	else			
1355	text = C_on ,			
1356	if (text != NULLSYM)			
1357	switch (name)			
1358	{ case C_missing_delays :			
1359	if (the -> missing_delays != NULLSYM)			
1360	error			
1361	(lkid ,			
1362	/*LMSC*/"report missing_delays respecified"			
1363) ,			
1364	else			
1365	the -> missing_delays = text ,			
1366	break ;			
1367	case C_io_store_changes :			
1368	if (the -> io_store_changes != NULLSYM)			
1369	error			
1370	(lkid ,			
1371	/*LMSC*/"report io_store_changes respecified"			
1372) ,			
1373	else			
1374	the -> io_store_changes = text ,			
1375	break ;			
1376	default : break ;			
1377	}			
1378	break ;			
1379	case M_MODELER :			
1380	tr_entry (fkid) ,			
1381	if (the -> mod_name == NULLSYM)			
1382	{			
1383	syntostr (textofnode (fkid)) != "" &&			
1384	syntostr (textofnode (fkid)) != "" &&			
1385	}			
1386	the -> dev_name = textofnode (fkid) ,			
1387	else			
1388	{ buf_string (syntostr (textofnode (fkid))) ,			
1389	the -> mod_name = strtosys (buf) ,			
1390	}			
1391	else			
1392	error			
1393	(fkid ,			
1394	/*LMSC*/"internal error: modeler_name respecified: %s"			
1395	, sof (fkid)			
1396) ,			
1397	tr_exit (fkid) ,			
1398	break ;			
1399	case M_CLOCK_TYPE :			
1400	tr_entry (fkid) ,			
1401	text = textofnode (fkid) ,			
1402	if (! compare (text , C_internal))			
1403	set_text (fkid , C_internal) ,			
1404	else if (! compare (text , C_ext1))			
1405	set_text (fkid , C_ext1) ,			
1406	else if (! compare (text , C_ext2))			
1407	set_text (fkid , C_ext2) ,			
1408	else			
1409	error			
1410	(fkid ,			
1411	/*LMSC*/"illegal clock_type specifier: %s (must be internal, external_clock0, or external_clock1)"			
1412	, syntostr (text)			
1413) ,			
1414	if (the -> clk_type == NULLSYM)			
1415	the -> clk_type = textofnode (fkid) ,			
1416	else			
1417	error			
1418	(fkid ,			
1419	/*LMSC*/"clock_type respecified: %s"			
1420	, syntostr (text)			
1421) ,			
1422	tr_exit (fkid) ,			
1423	break ;			
1424	case M_DIS_TIM_CHECK :			
1425	if (the -> dis_tim_check)			
1426	error			
1427	(T ,			
1428	/*LMSC*/"disable timing checking respecified"			
1429) ,			
1430	else			
1431	the -> dis_tim_check = 1 ;			
1432	break ;			
1433	case M_INH_TIM_MEASURE :			
1434	if (the -> inh_tim_measure)			
1435	error			
1436	(T ,			
1437	/*LMSC*/"inhibit timing measurement respecified"			
1438) ,			
1439	else			
1440	the -> inh_tim_measure = 1 ;			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE	5/23/89	PAGE #
TIME	4:42:25 pm	13/68

```

LINE # SOURCE TEXT
1441 break ;
1442 /*DEV statements*/
1443 case N_HOLD_TIME :
1444     if ( the -> def_hold == UNDEFINED )
1445     { the -> def_hold = 1000 * check_num ( fkid ) ;
1446       if ( the -> def_hold > MAX_PERIOD / 4 )
1447       { ( void ) printf
1448         ( "num_buf\n",
1449           "alu",
1450           ( unsigned long ) MAX_PERIOD / 4000
1451         ) ;
1452         error
1453         ( fkid ,
1454           /*MSG*/"device_hold_time too large: %s (must be no more than %s ns)"
1455           , syntostr ( textofnode ( fkid ) ) ,
1456             num_buf
1457         ) ;
1458     }
1459     else
1460     { error
1461       ( fkid ,
1462         /*MSG*/"device_hold_time respecified: %s"
1463         , syntostr ( textofnode ( fkid ) )
1464       ) ;
1465     }
1466     break ;
1467 case N_SETUP_TIME :
1468     if ( the -> def_setup == UNDEFINED )
1469     { the -> def_setup = 1000 * check_num ( fkid ) ;
1470       if ( the -> def_setup > MAX_PERIOD / 2 )
1471       { ( void ) printf
1472         ( "num_buf\n",
1473           "alu",
1474           ( unsigned long ) MAX_PERIOD / 2000
1475         ) ;
1476         error
1477         ( fkid ,
1478           /*MSG*/"device_setup_time too large: %s (must be no more than %s ns)"
1479           , syntostr ( textofnode ( fkid ) ) ,
1480             num_buf
1481         ) ;
1482     }
1483     else
1484     { error
1485       ( fkid ,
1486         /*MSG*/"device_setup_time respecified: %s"
1487         , syntostr ( textofnode ( fkid ) )
1488       ) ;
1489     }
1490     break ;
1491 case N_SAMPLE_TIME :
1492     if ( the -> def_sample == UNDEFINED )
1493     { the -> def_sample = 1000 * check_num ( fkid ) ;
1494       if ( the -> def_sample == 0 )
1495       { error
1496         ( fkid ,
1497           /*MSG*/"zero device_sample_time not allowed"
1498         ) ;
1499     }
1500     else
1501     { error
1502       ( fkid ,
1503         /*MSG*/"device_sample_time respecified: %s"
1504         , syntostr ( textofnode ( fkid ) )
1505       ) ;
1506     }
1507     break ;
1508 case N_SPEED :
1509     if ( kidscount ( fkid ) == 1 )
1510     { name = textofnode ( subtree ( 1 , fkid ) ) ;
1511       if
1512       ( "syntostr ( name ) >= 'A' && "syntostr ( name ) <= 'Z' ||
1513         "syntostr ( name ) >= 'a' && "syntostr ( name ) <= 'z'
1514       )
1515       { if ( scompare ( name , C_infinity ) )
1516         { set_text ( subtree ( 1 , fkid ) , C_infinity ) ;
1517           name = C_as ;
1518         }
1519         else if ( scompare ( name , C_static ) )
1520         { set_text ( subtree ( 1 , fkid ) , C_static ) ;
1521           name = C_Hz ;
1522         }
1523         else
1524         { error
1525           ( subtree ( 1 , fkid ) ,
1526             /*MSG*/"illegal device_speed identifier: %s (must be infinity or static)"
1527           , syntostr ( name )
1528         ) ;
1529         name = NULLSYM ;
1530       }
1531       rate1 = rate2 = 0 ;
1532       if ( kids == 2 )
1533       { text = textofnode ( subtree ( 1 , lkid ) ) ;
1534         if
1535         ( "syntostr ( text ) >= 'A' &&
1536           "syntostr ( text ) <= 'Z' ||
1537           "syntostr ( text ) >= 'a' &&
1538           "syntostr ( text ) <= 'z'
1539         )
1540         { if ( scompare ( text , C_infinity ) )
1541           { set_text ( subtree ( 1 , lkid ) , C_infinity ) ;
1542             text = C_as ;
1543           }
1544           else if ( scompare ( text , C_static ) )
1545           { set_text ( subtree ( 1 , lkid ) , C_static ) ;
1546             text = C_Hz ;
1547           }
1548           else
1549           { error
1550             ( subtree ( 1 , lkid ) ,
1551               /*MSG*/"illegal device_speed identifier: %s (must be infinity or static)"
1552             , syntostr ( text )
1553           ) ;
1554             text = NULLSYM ;
1555           }
1556         }
1557         else
1558         { rate2 = do_unit ( lkid , &text ) ;
1559         }
1560     }
1561     else

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_1.c	DATE 5/23/89	PAGE # 14/69
LINE #	SOURCE TEXT			
1561	{ rate1 = rate2 = do_unit (lkid , tname) ;			
1562	if (kids == 2)			
1563	{ if (kidcount (lkid) == 1)			
1564	{ text = textofnode (subtree (1 , lkid)) ;			
1565	if			
1566	{ syntostr (text) >= 'A' &&			
1567	syntostr (text) <= 'Z' &&			
1568	syntostr (text) >= 'a' &&			
1569	syntostr (text) <= 'z' &&			
1570	{ if (acompare (text , C_infinity))			
1571	{ set_text			
1572	{ subtree (1 , lkid) ,			
1573	C_infinity			
1574	}			
1575	text = C_ms ;			
1576	}			
1577	else if (acompare (text , C_static))			
1578	{ set_text			
1579	{ subtree (1 , lkid) ,			
1580	C_static			
1581	}			
1582	text = C_Hz ;			
1583	}			
1584	else			
1585	{ error			
1586	{ subtree (1 , lkid) ,			
1587	syntostr (text)			
1588	}			
1589	/*MSG*/"illegal device_speed identifier: ts (must be infinity or static)"			
1590	}			
1591	text = NULLSYM ;			
1592	}			
1593	rate2 = 0 ;			
1594	}			
1595	else			
1596	rate2 = do_unit (lkid , ttext) ;			
1597	}			
1598	else			
1599	rate2 = do_unit (lkid , ttext) ;			
1600	}			
1601	}			
1602	else			
1603	{ rate1 = rate2 = do_unit (lkid , tname) ;			
1604	if (kids == 2)			
1605	{ if (kidcount (lkid) == 1)			
1606	{ text = textofnode (subtree (1 , lkid)) ;			
1607	if			
1608	{ syntostr (text) >= 'A' &&			
1609	syntostr (text) <= 'Z' &&			
1610	syntostr (text) >= 'a' &&			
1611	syntostr (text) <= 'z' &&			
1612	{ if (acompare (text , C_infinity))			
1613	{ set_text (subtree (1 , lkid) , C_infinity) ;			
1614	text = C_ms ;			
1615	}			
1616	else if (acompare (text , C_static))			
1617	{ set_text (subtree (1 , lkid) , C_static) ;			
1618	text = C_Hz ;			
1619	}			
1620	else			
1621	{ error			
1622	{ subtree (1 , lkid) ,			
1623	syntostr (text)			
1624	}			
1625	/*MSG*/"illegal device_speed identifier: ts (must be infinity or static)"			
1626	}			
1627	text = NULLSYM ;			
1628	}			
1629	rate2 = 0 ;			
1630	}			
1631	else			
1632	rate2 = do_unit (lkid , ttext) ;			
1633	}			
1634	else			
1635	rate2 = do_unit (lkid , ttext) ;			
1636	}			
1637	}			
1638	if (kids == 2)			
1639	switch (text)			
1640	{ case C_Hz :			
1641	case C_KHz :			
1642	case C_MHz :			
1643	case C_GHz :			
1644	{ switch (name)			
1645	{ case C_ms :			
1646	case C_us :			
1647	case C_ns :			
1648	case C_ps :			
1649	buf_and (name , text) ;			
1650	error			
1651	{ subtree (kidcount (lkid) , lkid) ,			
1652	buf			
1653	/*MSG*/"inconsistent unit types: ts"			
1654	}			
1655	break ;			
1656	default :			
1657	break ;			
1658	}			
1659	break ;			
1660	case C_ms :			
1661	case C_us :			
1662	case C_ns :			
1663	case C_ps :			
1664	{ switch (name)			
1665	{ case C_Hz :			
1666	case C_KHz :			
1667	case C_MHz :			
1668	case C_GHz :			
1669	buf_and (name , text) ;			
1670	error			
1671	{ subtree (kidcount (lkid) , lkid) ,			
1672	buf			
1673	/*MSG*/"inconsistent unit types: ts"			
1674	}			
1675	break ;			
1676	default :			
1677	break ;			
1678	}			
1679	}			
1680	break ;			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_1.c

DATE 5/23/89
TIME 4:42:25 pm

PAGE #
15/70

```

LINE #          SOURCE TEXT
1681      default : break ;
1682      }
1683      if ( ratel == 0 && rate2 == 0 )
1684      {
1685          error
1686          ( "id",
1687            /*"no maximum device speed specified"
1688            */
1689            if ( ratel == 0 || rate2 != 0 && ratel > rate2 )
1690            {
1691                tmp = ratel ;
1692                ratel = rate2 ;
1693                rate2 = tmp ;
1694            }
1695      # undef ENFORCE_RATE
1696      # ifdef ENFORCE_RATE
1697          ( "void" ) sprintf
1698          ( "max",
1699            "1u to 1u",
1700            MIN_PERIOD / 1000 ,
1701            MAX_PERIOD / 1000
1702          ) ;
1703      #endif
1704      error
1705      ( "device speed out of supported range (%s ns)"
1706        , "max"
1707      ) ;
1708      # undef ENFORCE_RATE
1709      if ( the -> clk_period1 == -1 )
1710      {
1711          the -> clk_period1 = ratel ;
1712          the -> clk_period2 = rate2 ;
1713      }
1714      else
1715      {
1716          error
1717          ( "id",
1718            /*"MSG"/"device speed respecified"
1719            */
1720            break ;
1721            case N_TECHNOLOGY :
1722                tr_entry ( "id" ) ;
1723                text = textofnode ( "id" ) ;
1724                if ( strcmp ( text , C_TTL ) )
1725                {
1726                    else if ( strcmp ( text , C_NMOS ) )
1727                    {
1728                        set_text ( "id", C_NMOS ) ;
1729                    }
1730                    else if ( strcmp ( text , C_CMOS ) )
1731                    {
1732                        set_text ( "id", C_CMOS ) ;
1733                    }
1734                    else if ( strcmp ( text , C_LCMOS ) )
1735                    {
1736                        set_text ( "id", C_LCMOS ) ;
1737                    }
1738                    else if ( strcmp ( text , C_CUSTOM ) )
1739                    {
1740                        set_text ( "id", C_CUSTOM ) ;
1741                    }
1742                    else
1743                    {
1744                        error
1745                        ( "id",
1746                          /*"MSG"/"illegal technology: %s (must be TTL, NMOS, CMOS, or LCMOS)"
1747                          */
1748                          , systestr ( text )
1749                        ) ;
1750                        if ( the -> technology == NULLSYM )
1751                        {
1752                            the -> technology = textofnode ( "id" ) ;
1753                            the -> tech_tree = T ;
1754                        }
1755                        else
1756                        {
1757                            error
1758                            ( "id",
1759                              /*"MSG"/"technology respecified: %s"
1760                              */
1761                              , systestr ( text )
1762                            ) ;
1763                            tr_exit ( "id" ) ;
1764                            for ( k = 2 ; k <= kids ; ++k )
1765                                do_tech_spec ( subtree ( k , T ) ) ;
1766                            break ;
1767                            case N_INITIALIZE :
1768                                for ( k = 1 ; k <= kids ; ++k )
1769                                    the -> mem_ops -> do_resets ( subtree ( k , T ) ) ;
1770                                break ;
1771                            /*"more .DEV statements"/
1772                            case N_IN_PINS :
1773                                for ( k = 1 ; k <= kids ; ++k )
1774                                    do_in_pins ( subtree ( k , T ) ) ;
1775                                break ;
1776                            case N_IO_PINS :
1777                                for ( k = 1 ; k <= kids ; ++k )
1778                                    do_io_pins ( subtree ( k , T ) ) ;
1779                                break ;
1780                            case N_NC_PINS :
1781                                for ( k = 1 ; k <= kids ; ++k )
1782                                    do_nc_pins ( subtree ( k , T ) ) ;
1783                                break ;
1784                            case N_OUT_PINS :
1785                                for ( k = 1 ; k <= kids ; ++k )
1786                                    do_out_pins ( subtree ( k , T ) ) ;
1787                                break ;
1788                            case N_POWER_PINS :
1789                                for ( k = 1 ; k <= kids ; ++k )
1790                                    do_power_pins ( subtree ( k , T ) ) ;
1791                                break ;
1792                            case N_GROUND_PINS :
1793                                for ( k = 1 ; k <= kids ; ++k )
1794                                    do_ground_pins ( subtree ( k , T ) ) ;
1795                                break ;
1796                            /*".TMC statements"/
1797                            case N_DEFAULT_DELAY :
1798                                process_timing
1799                                ( "id",
1800                                  & the -> min_default ,
1801                                  & the -> typ_default ,
1802                                  & the -> max_default
1803                                ) ;
1804                                if ( the -> use_default )
1805                                {
1806                                    error
1807                                    ( "id",
1808                                      /*"MSG"/"default delay respecified"
1809                                      */
1810                                      ,
1811                                      else
1812                                      the -> use_default = 1 ;
1813                                      break ;
1814                                }
1815                                case N_DELAY :
1816                                case N_PACKAGE :
1817                                case N_PIN_MAP :
1818                                break ; /* get these in a later pass */

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_1.c	DATE 5/23/89	PAGE # 16/71
			TIME 4:42:25 pm	
LINE #	SOURCE TEXT			
1801	case N ADAPTER :			
1802	the -> has_adapter_map = 1 ,			
1803	for (k = 1 , k <= kids , ++k)			
1804	do_mapping (subtree (k , T) , T) ;			
1805	break ;			
1806	default :			
1807	if (kids)			
1808	while (kidcount (fkid))			
1809	fkid = subtree (1 , fkid) ;			
1810	else			
1811	fkid = T ,			
1812	error			
1813	(fkid ,			
1814	/*EMSG*/"internal error: walk_1: unknown node name: %s"			
1815	, systostr (name)			
1816) ;			
1817	break ;			
1818	}			
1819	tr_exit (T) ;			
1820	} /* walk_1 */			
1821				
1822	/* end of Constrainer (walk_1) for BML Processor */			
1823				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_2.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
1/72

```

1  /* SCCS ID: walk_2.c rev 3.1.1, 4/24/89 at 08:01:43 */
2
3  /*
4   * Constrainer (walk_2) for HGL Processor
5   */
6
7  #include "common.h"
8  #include "HGL.h"
9  #include <strings.h>
10 #include <string.h>
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <unistd.h>
14
15 static void
16 mapit ( from , to , parent , grandparent )
17 {
18     from ,
19     to ,
20     parent ,
21     grandparent ;
22     /* process a single mapping of the package mapping */
23     {
24         look ,
25         dcla ,
26         symbol ,
27         text1 ,
28         text2 ;
29         tr_entry ( from ) ;
30         if ( nameofnode ( from ) == P_STR )
31             ( buf_string ( symtostr ( textofnode ( from ) ) ) ) ;
32         text1 = strtosym ( buf ) ;
33     }
34     else
35         text1 = textofnode ( from ) ;
36         dcla = NULLTREE ;
37         if ( ( look = l_lookup ( text1 ) ) != NULLTREE &&
38             ( nameofnode ( look ) == N_PACKAGE ||
39               nameofnode ( look ) == N_MAPPING )
40             )
41             )
42             )
43             error
44             ( from ,
45             /*MSG*/"package mapping respecified for package pin name: %s"
46             , symtostr ( text1 )
47             )
48         else
49             dcla = grandparent ;
50             tr_exit ( from ) ;
51             tr_entry ( to ) ;
52             text2 = textofnode ( to ) ;
53             if ( ( look = g_lookup ( text2 ) ) != NULLTREE ||
54                 nameofnode ( look ) != N_ADAPTER
55                 )
56                 )
57                 if ( dcla != NULLTREE )
58                     dcla = parent ;
59                 if ( dcla != NULLTREE )
60                     l_declare ( text1 , dcla ) ;
61                 if ( spin_name_of ( text2 ) )
62                     error
63                     ( to ,
64                     /*MSG*/"duplicate device adapter pin name: %s"
65                     , symtostr ( text2 )
66                     )
67                 else
68                     set_spin_name ( text2 , text1 ) ;
69                 tr_exit ( to ) ;
70             } /* mapit */
71 }
72
73 static void
74 do_mapping ( T , parent )
75 {
76     T ,
77     parent ;
78     /* process the package mappings */
79     {
80         kid1 ,
81         kid2 ,
82         ushort ,
83         kid1 ,
84         kid2 ;
85         tr_entry ( T ) ;
86         kid1 = subtree ( 1 , T ) ;
87         tr_entry ( kid1 ) ;
88         kid1 = kidcount ( kid1 ) ;
89         kid2 = subtree ( 2 , T ) ;
90         kid2 = kidcount ( kid2 ) ;
91         if ( kid1 > kid2 )
92             if ( buf_instead ( kid2 , kid1 ) )
93                 error
94                 ( subtree ( kid2 , kid2 ) ,
95                 /*MSG*/"too few device adapter pin names: %s"
96                 , buf
97                 )
98             else if ( kid1 < kid2 )
99                 error
100                 ( subtree ( kid1 + 1 , kid2 ) ,
101                 /*MSG*/"too many device adapter pin names: %s"
102                 , symtostr ( textofnode ( subtree ( kid1 + 1 , kid2 ) ) )
103                 )
104             )
105             tr_exit ( kid1 ) ;
106             tr_entry ( kid2 ) ;
107             for ( k = 1 , k <= kid1 , ++k )
108                 if ( k <= kid2 )
109                     mapit ( subtree ( k , kid1 ) , subtree ( k , kid2 ) , T , parent ) ;
110             tr_exit ( kid2 ) ;
111             tr_exit ( T ) ;
112         } /* do_mapping */
113 }
114
115 void
116 walk_2 ( T )
117 {
118     tree T ;
119     /* the constraints second walk */
120     {
121         ushort
122         k
123         kids ;

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_2.c	DATE, 5/23/89 TIME 4:42:27 pm	PAGE # 2/73
LINE #	SOURCE TEXT			
121	tr_entry (T) ,			
122	kids = kidcount (T) ;			
123	switch (sameofnode (T))			
124	{ case N_PACKAGE :			
125	the -> has_package_map = 1 ;			
126	for (k = 1 ; k <= kids ; ++k)			
127	do_mapping (subtree (k , T) , T) ;			
128	break ;			
129	default : break ;			
130	}			
131	tr_exit (T) ,			
132	} /* walk_2 */			
133				
134	/* end of Constrainer {walk_2} for RMBL Processor */			
135				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_3.c

DATE	5/23/89	PAGE #
TIME	4:42:27 pm	1/74

```

1  /* SCCS ID: walk_3.c rev 3.1, 4/24/89 at 08:01:46 */
2  /*
3  *
4  * .Constrainer (walk_3) for EMBL Processor
5  *
6  */
7  #include "common.h"
8  #include "EMBL.h"
9  #include "strings.h"
10 #include "string.h"
11 #include "sparse.h"
12 #include "trees.h"
13 #include "hconst.h"
14
15 static struct pin_attr *
16 allo_attrs ( size )
17 unsigned size
18 /* returns a pointer to a newly-allocated pin attribute array */
19 {
20     char *ptr;
21     if ( size )
22         ptr = malloc ( sizeof ( struct pin_attr ) * size );
23     else
24         return NULL;
25     if ( ptr == NULL )
26         {
27             /*LMSC*/out of memory on modular (allo_attrs)
28             return NULL;
29             /*LMSC*/
30         }
31     return ( struct pin_attr * ) ptr;
32 } /* allo_attrs */
33
34 static void
35 declare_pin_name ( text, T, direction, overdeclare )
36 symbol text,
37 tree T,
38 direction,
39 unsigned overdeclare;
40 /* declare passed pin name */
41 {
42     tree look;
43     if ( ! overdeclare && ( look = g_lookup ( text ) ) != NULLTREE )
44     {
45         nameofnode ( look ) == N_IO_PINS ||
46         nameofnode ( look ) == N_NC_PINS ||
47         nameofnode ( look ) == N_OUT_PINS ||
48         nameofnode ( look ) == N_POWER_PINS ||
49         nameofnode ( look ) == N_GROUND_PINS
50     }
51     }
52     error
53     ( T,
54     /*LMSC*/"Device signal name redeclared: %s"
55     , syntostr ( text )
56     );
57     else
58     {
59         g_declare ( text, direction );
60     } /* Declare_pin_name */
61
62 static unsigned
63 add_pin_name ( T, direction, ndex, dest, overdeclare )
64 tree T,
65 direction,
66 unsigned ndex;
67 struct pins *dest;
68 unsigned overdeclare;
69 /* adds the pin name(s) in T starting at the ndex'th position in dest */
70 /* declares them all via declare_pin_name(). returns how many pins */
71 {
72     unsigned
73     i,
74     sum1 = 0,
75     sum2 = 0;
76     symbol text;
77     tree kid, look;
78     tr_entry ( T );
79     switch ( nameofnode ( T ) )
80     {
81         case P_ID:
82             text = textofnode ( T );
83             dest->name = text;
84             declare_pin_name
85             ( dest->name,
86             T,
87             direction,
88             overdeclare
89             );
90             if ( ! overdeclare )
91                 set_pin_name ( dest->name, ndex );
92             break;
93         case P_STR:
94             /* what about generators in strings? */
95             buf_string ( syntostr ( textofnode ( T ) ) );
96             text = strtosym ( buf );
97             declare_pin_name
98             ( dest->name = text,
99             T,
100             direction,
101             overdeclare
102             );
103             if ( ! overdeclare )
104                 set_pin_name ( dest->name, ndex );
105             break;
106         case N_PIN_NAME:
107             kid = subtree ( 1, T );
108             tr_entry ( kid );
109             text = textofnode ( kid );
110             look = g_lookup ( text );
111             if ( look == NULLTREE )
112                 g_declare ( text, T );
113             /* add buss name to a list of buss names? */
114             tr_exit ( kid );
115             kid = subtree ( 2, T );
116             tr_entry ( kid );
117             sum1 =
118             dec_ration ( kid );
119             tr_exit ( kid );
120

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_3.c	DATE 5/23/89	PAGE # 2/75
LINE #		SOURCE TEXT		
121		if (kidcount (T) == 3)		
122		{ kid = subtree (3 , T) ;		
123		tr_entry (kid) ;		
124		num1 = dec_ratio (kid) ;		
125		tr_exit (kid) ;		
126		}		
127		if (num1 > num2)		
128		for (i = num1 ; --i)		
129		{ declare_pin_name		
130		(dest -> name = gen_name (text , i) ,		
131		T ,		
132		direction ,		
133		overdeclare		
134) ;		
135		if (! overdeclare)		
136		set_pin_name (dest -> name , ndex++) ;		
137		/* record highest bus number in this bus? */		
138		if (i == num1)		
139		break ;		
140		++dest ;		
141		}		
142		else		
143		for (i = num1 ; ++i)		
144		{ declare_pin_name		
145		(dest -> name = gen_name (text , i) ,		
146		T ,		
147		direction ,		
148		overdeclare		
149) ;		
150		if (! overdeclare)		
151		set_pin_name (dest -> name , ndex++) ;		
152		/* record highest bus number in this bus? */		
153		if (i == num1)		
154		break ;		
155		++dest ;		
156		}		
157		break ;		
158		default :		
159		error		
160		(T ,		
161		/*MSG*/"internal error: add_pin_name: unknown node name: %s"		
162		, syntostr (nameofnode (T))		
163) ;		
164		break ;		
165		}		
166		tr_exit (T) ;		
167		return num1 > num2 ? num1 - num1 + 1 : num2 - num1 + 1 ;		
168		} /* add_pin_name */		
169		}		
170		static void		
171		do_pins (T , direction)		
172		{		
173		T ,		
174		direction ,		
175		/* do all the pins in a pin correspondence via add_pin_name() */		
176		/* add attributes to them all as well */		
177		{ ushort		
178		i ,		
179		j ,		
180		kids ,		
181		k ,		
182		count = 0 ,		
183		start ,		
184		num ,		
185		tree		
186		{		
187		kid ,		
188		kid2 ,		
189		look ,		
190		struct pin_attr *attrs ,		
191		symbol text ;		
192		start = the -> pin_cnt ;		
193		tr_entry (T) ;		
194		kids = kidcount (T) ;		
195		for		
196		{ kid = subtree (i - 1 , T) ;		
197		nameofnode (kid) != N_NAMES ,		
198		kid = subtree (++i , T)		
199		}		
200		{ if ((num = dec_ratio (kid)) == 0)		
201		num = 1 ;		
202		k = num ;		
203		while (k--)		
204		{ j =		
205		add_pin_name		
206		(kid ,		
207		direction ,		
208		the -> pin_cnt ,		
209		the -> pin_cnt + the -> pin_cnt ,		
210		(ushort) (k != num - 1)		
211) ;		
212		count += j ;		
213		the -> pin_cnt += j ;		
214		}		
215		ASSERT(start+count==the->pin_cnt);		
216		tr_exit (kid) ;		
217		k = kidcount (kid) ;		
218		for (j = 0 ; j < count ; ++j)		
219		{ if (j < k)		
220		{ text = textofnode (kid2 = subtree (j + 1 , kid)) ;		
221		if (nameofnode (kid2) == P_STR)		
222		{ buf_string (syntostr (text)) ;		
223		text = strtosym (buf) ;		
224		}		
225		if		
226		{ (look = l_lookup (text)) == NULLTREE		
227		nameofnode (look) != N_PACKAGE		
228		}		
229		if (look != NULLTREE && nameofnode (look) == N_MAPPING)		
230		{ if (the -> has_adapter_map)		
231		error		
232		(kid2 ,		
233		/*MSG*/"package pin name %s has no device adapter mapping"		
234		, syntostr (text)		
235) ;		
236		}		
237		else if (the -> has_package_map)		
238		error		
239		(kid2 ,		
240		/*MSG*/"no such package pin name: %s"		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_3.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
3/76

```

LINE #          SOURCE TEXT
241          , symtostr ( text )
242          ) ,
243          if ( pin_number_of ( text ) )
244              error
245              ( kid2 ,
246              /*MSG*/"duplicate package ... name: %s"
247              , symtostr ( text )
248              ) ,
249          else
250              { set_pin_number ( text , the -> pins [ start + j ] . name ) ,
251                the -> pins [ start + j ] . pkg_name = text ;
252              }
253          }
254          tr_exit ( kid ) ;
255          attr = allo_attr ( sum - kids - ( j - 1 ) ) ;
256          while ( ++i < kids )
257              { kid = subtree ( i , T ) ;
258                add_attr ( kid , attr , i - j - 1 ) ;
259              }
260          while ( start < the -> pin_cat )
261              { the -> pins [ start ] . alias = NULLSYM ;
262                the -> pins [ start ] . number = MAX_PINS ;
263                the -> pins [ start ] . trace_delay = 0 ;
264                the -> pins [ start ] . sum_attr = sum ;
265                the -> pins [ start ] . attr = attr ;
266                ++start ;
267              }
268          tr_exit ( T ) ;
269          } /* do_pins */
270
271          void
272          walk_3 ( T )
273          { tree T ;
274            /* the constraints third walk */
275            { ushort
276              k ,
277              kids ;
278              tr_entry ( T ) ;
279              kids = kidcount ( T ) ;
280              switch ( nameofnode ( T ) )
281              { case N_IN_PINS :
282                case N_IO_PINS :
283                case N_MC_PINS :
284                case N_OUT_PINS :
285                case N_POWER_PINS :
286                case N_GROUND_PINS :
287                  for ( k = 1 ; k <= kids ; ++k )
288                      do_pins ( subtree ( k , T ) , T ) ;
289                  break ;
290              default :
291                  break ;
292              }
293              tr_exit ( T ) ;
294            } /* walk_3 */
295          }
296          /* end of Constraints (walk_3) for SBL Processor */
297

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

TIME

4:42:27 pm

1/77

```

1  /* SCES_ID: walk_4.c rev 3.1.1, 4/24/89 at 08:01:53 */
2
3  /*
4   * Constraint (walk_4) for HSM1 Processor
5   */
6
7  #include "common.h"
8  #include "HSM1.h"
9  #include "strings.h"
10 #include "stray.h"
11 #include "hparam.h"
12 #include "trees.h"
13 #include "hconst.h"
14
15 static char num_buf [ 32 ] , /* for error reporting */
16
17 static struct timing *
18 allo_timing ( )
19 /* returns a pointer to a newly-allocated timing structure */
20 {
21     char *ptr ;
22     ptr = malloc ( sizeof ( struct timing ) ) ;
23     if ( ptr == NULL )
24     {
25         perror ( "out of memory on modular (allo_timing)" ) ;
26         return ( 0 ) ;
27     }
28     return ( struct timing * ) ptr ;
29 } /* allo_timing */
30
31 ushort
32 has_feedback_attribute ( text )
33 symbol text ;
34 /* returns 1 iff the passed pin name has the feedback attribute */
35 {
36     ushort
37     i ,
38     ndex ,
39     ndex = pin_name_of ( text ) ;
40     for ( i = 0 ; i < the->pins [ ndex ] . num_attr ; ++i )
41     if ( the->pins [ ndex ] . attr [ i ] . use_flag == ID_ATTR )
42     {
43         text = the->pins [ ndex ] . attr [ i ] . attr . id ;
44         if ( text == C_feedback )
45             return 1 ;
46     }
47     return 0 ;
48 } /* has_feedback_attribute */
49
50 static void
51 do_reset_pin ( T , text , dest , offset )
52 tree T ;
53 symbol text ;
54 struct sequence *dest ;
55 ushort offset ;
56 /* adds the sequence pin name into dest at offset */
57 {
58     ushort i ;
59     tree look ;
60     look = g_lookup ( text ) ;
61     if ( look == NULLTREE )
62     {
63         perror ( "no such device signal name: %s" ,
64                 systr ( text ) ) ;
65     }
66     else
67     switch ( nameofnode ( look ) )
68     {
69         case M_IN_PINS :
70             break ;
71         case M_IO_PINS :
72             break ;
73         case M_OUT_PINS :
74             if ( ! has_feedback_attribute ( text ) )
75                 perror ( "device signal name %s does not have the feedback attribute" ,
76                         systr ( text ) ) ;
77             break ;
78         case M_NC_PINS :
79             break ;
80         case M_POWER_PINS :
81             break ;
82         case M_GROUND_PINS :
83             break ;
84         default :
85             perror ( "%s is not a device signal name" ,
86                     systr ( text ) ) ;
87             break ;
88     }
89     for ( i = 0 ; i < offset ; ++i )
90     if ( dest [ i ] . pin_name == text )
91     {
92         perror ( "%s is not a device signal name" ,
93                 systr ( text ) ) ;
94         break ;
95     }
96     /* initialization sequence for device signal name is reestablished */
97     dest [ offset ] . pin_name = text ;
98     /* do_reset_pin */
99
100 static ushort
101 add_reset_pin ( T , dest , offset )
102 tree T ;
103 struct sequence *dest ;
104 ushort offset ;
105 /* calls do_reset_pin() for each pin indicated by T */
106 {
107     ushort
108     i ,
109     sum1 = 0 ,
110     sum2 = 0 ;
111     symbol text ;
112     tree kid ;
113     tr_entry ( T ) ;
114     switch ( nameofnode ( T ) )

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

TIME 4:42:27 pm

2/78

```

121 LINE # SOURCE TEXT
122 { case P_ID :
123     case P_NUM :
124         do_reset_pis ( T , textofnode ( T ) , dest , offset ) ,
125         break ;
126     case P_STR :
127         about_generators is strings? */
128         buf_string ( syntostr ( textofnode ( T ) ) ) ;
129         text = strtoyma ( buf ) ;
130         do_reset_pis ( T , text , dest , offset ) ,
131         break ;
132     case N_FIN_NAME :
133         kid = subtree ( 1 , T ) ;
134         tr_entry ( kid ) ;
135         text = textofnode ( kid ) ;
136         tr_exit ( kid ) ;
137         kid = subtree ( 2 , T ) ;
138         tr_entry ( kid ) ;
139         sum1 =
140             decoration ( kid ) ;
141         tr_exit ( kid ) ;
142         if ( kidcount ( T ) == 1 )
143             [ kid = subtree ( 3 , T ) ;
144               tr_entry ( kid ) ;
145               sum1 = decoration ( kid ) ;
146               tr_exit ( kid ) ;
147             ]
148         if ( sum1 > sum2 )
149             for ( i = sum1 ; i-- ; )
150                 [ do_reset_pis ( T , gen_name ( text , i ) , dest , offset ) ;
151                   if ( i == sum1 )
152                       break ;
153                   ++offset ;
154                 ]
155         else
156             for ( i = sum1 ; i-- ; )
157                 [ do_reset_pis ( T , gen_name ( text , i ) , dest , offset ) ;
158                   if ( i == sum1 )
159                       break ;
160                   ++offset ;
161                 ]
162         break ;
163     default :
164         error
165             ( T ,
166             /*LMSG*/"internal error: add_reset_pis: unknown node name: %s"
167             , syntostr ( nameofnode ( T ) )
168             ) ;
169         break ;
170     }
171     tr_exit ( T ) ;
172     return sum1 > sum2 ? sum1 - sum2 + 1 : sum2 - sum1 + 1 ;
173 } /* add_reset_pis */
174
175 static unsigned long
176 upseq_len ( T )
177     tree T ;
178 /* recursively determines the length of an up-sequence */
179 { ushort
180     k ,
181     kids ,
182     unsigned long len ,
183     symbol text ,
184     tree kid ;
185     switch ( nameofnode ( T ) )
186     { case P_NUM :
187         len = check_int ( T ) ;
188         if ( len < 1 || len >= 1 )
189             error
190                 ( T ,
191                 /*LMSG*/"illegal initialization sequence value (must be 0 or 1): %s"
192                 , sof ( T )
193                 ) ;
194         decorate ( T , ( long ) ( len == 1 ) ) ;
195         return 1 ;
196     case N_LIST :
197         tr_entry ( T ) ;
198         len = 0 ;
199         kids = kidcount ( T ) ;
200         for ( k = 1 ; k <= kids ; ++k )
201             len += upseq_len ( subtree ( k , T ) ) ;
202         tr_exit ( T ) ;
203         return len ;
204     case N_REPEAT :
205         tr_entry ( T ) ;
206         len = check_int ( subtree ( 1 , T ) ) ;
207         if ( len == 0 )
208             error
209                 ( subtree ( 1 , T ) ,
210                 /*LMSG*/"zero repeat factor in initialize sequence illegal"
211                 ) ;
212         if ( len > MAX_RES_SEQ_LEN )
213             [ ( void ) sprintf ( buf , "%lu" , len ) ;
214               ( void ) sprintf
215                   ( sum_buf ,
216                   "%lu"
217                   , ( unsigned long ) MAX_RES_SEQ_LEN
218                   ) ;
219               error
220                   ( subtree ( 1 , T ) ,
221                   /*LMSG*/"initialize sequence repeat factor too large: %s (must be no more than %s)"
222                   , buf ,
223                   sum_buf
224                   ) ;
225               len = 0 ;
226             ]
227         decorate ( subtree ( 1 , T ) , ( long ) len ) ;
228         len += upseq_len ( subtree ( 2 , T ) ) ;
229         tr_exit ( T ) ;
230         return len ;
231     case N_FEEDBACK :
232         tr_entry ( T ) ;
233         kid = subtree ( 1 , T ) ;
234         tr_entry ( kid ) ;
235         text = textofnode ( kid ) ;
236         if ( acompare ( text , C_rise ) )
237             set_text ( kid , C_rise ) ;
238         else if ( acompare ( text , C_fall ) )
239             set_text ( kid , C_fall ) ;
240         else

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

TIME

4:42:27 pm

3/79

```

LINE # SOURCE TEXT
241      cerrror
242      ( kid ,
243      /*LMSC*/"illegal feedback condition: %s (must be rise or fall)"
244      , syntostr ( text )
245      ) ,
246      cerrror
247      ( kid ,
248      /*LMSC*/"feedback sequences illegal here"
249      ) ,
250      tr_exit ( kid ) ,
251      len = upseq_len ( subtree ( 2 , T ) ) ,
252      tr_exit ( T ) ,
253      return len ,
254      default :
255      lm_error
256      ( 1
257      /*LMSC*/"internal error: upseq_len: bad node name: %s"
258      , syntostr ( nameofnode ( T ) )
259      ) ,
260      return 0 ,
261      } /* upseq_len */
262
263
264 static void
265 do_reset_seq
266 ( T
267   is_rise_addr ,
268   pre_length_addr ,
269   length_addr ,
270   post_length_addr
271 )
272 {
273   tree T ;
274   ushort
275     *is_rise_addr ,
276     *pre_length_addr ,
277     *length_addr ,
278     *post_length_addr ;
279   /* processes a reset sequence specification */
280   {
281     k ,
282     k2 ,
283     count ,
284     kids ,
285     kids2 ,
286     start ,
287     fb_seqn = 0 ,
288     tree
289     kid ,
290     kid2 ,
291     kid3 ,
292     loop
293     len ,
294     pre_len = 0 ,
295     post_len = 0 ,
296     symbol_text ,
297     count =
298     start =
299     the -> reset_cnt ,
300     *is_rise_addr =
301     *length_addr =
302     0 ,
303     tr_entry ( T ) ,
304     kids = kidcount ( T ) ,
305     for ( k = 1 , k < kids , ++k )
306       ( kid = subtree ( k , T ) ,
307         count += add_reset_pin ( kid , the -> sequences , count ) ,
308       )
309     the -> reset_cnt = count ,
310     kid = subtree ( kids , T ) ,
311     tr_entry ( kid ) ,
312     kids2 = kidcount ( kid ) ,
313     for ( k2 = 1 , k2 < kids2 , ++k2 )
314       ( kid2 = subtree ( k2 , kid ) ,
315         switch ( nameofnode ( kid2 ) )
316         {
317           case P_REPEAT :
318             if ( fb_seqn )
319               ( post_len += upseq_len ( kid2 ) ,
320                 if ( post_len > MAX_RES_SEQ_LEN )
321                   ( void ) sprintf ( buf , "%ld" , post_len ) ,
322                   ( void ) sprintf ( mem_buf ,
323                     "%ld" ,
324                     ( unsigned long ) MAX_RES_SEQ_LEN
325                   ) ,
326                 cerrror
327                 ( kid2 ,
328                 /*LMSC*/"post-feedback initialization sequence too long: %s (must be no more than %s)"
329                 , buf ,
330                 mem_buf
331                 ) ,
332                 post_len = 0 ,
333               )
334             else
335               ( pre_len += upseq_len ( kid2 ) ,
336                 if ( pre_len > MAX_RES_SEQ_LEN )
337                   ( void ) sprintf ( buf , "%ld" , pre_len ) ,
338                   ( void ) sprintf ( mem_buf ,
339                     "%ld" ,
340                     ( unsigned long ) MAX_RES_SEQ_LEN
341                   ) ,
342                 cerrror
343                 ( kid2 ,
344                 /*LMSC*/"pre-feedback initialization sequence too long: %s (must be no more than %s)"
345                 , buf ,
346                 mem_buf
347                 ) ,
348                 pre_len = 0 ,
349               )
350             break ;
351           case N_FEEDBACK :
352             tr_entry ( kid2 ) ,
353             kid3 = subtree ( 1 , kid2 ) ,
354             tr_entry ( kid3 ) ,
355             text = textofnode ( kid3 ) ,
356             if ( scompare ( text , C_rise ) )
357

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

4/80

TIME 4:42:27 pm

```

LINE # SOURCE TEXT
361 set_text ( kid1, C_rise );
362 else if ( !compare ( text, C_fall ) )
363 set_text ( kid1, C_fall );
364 else
365 error
366 ( kid
367 /*IMSC*/"illegal feedback condition: ts (must be rise or fall)"
368 , syntostr ( text )
369 );
370 tr_exit ( kid1 );
371 len = upsize ( subtree ( 2, kid2 ) );
372 if ( len > MAX_FB_SEQ_LEN )
373 { ( void ) sprintf ( buf, "ld", len );
374 ( void ) sprintf
375 ( sum_buf,
376 "ulu",
377 ( unsigned long ) MAX_FB_SEQ_LEN
378 );
379 error
380 ( subtree ( 2, kid2 ),
381 /*IMSC*/"feedback sequence too long: ts (must be no more than ts)"
382 , buf,
383 sum_buf
384 );
385 len = 0;
386 }
387 if ( fb_seq )
388 error
389 ( subtree ( 1, kid2 ),
390 /*IMSC*/"feedback sequence respecified"
391 );
392 else
393 { fb = k2;
394 fb_seq = 1;
395 *is_rise_addr = textofnode ( kid1 ) == C_rise;
396 *length_addr = len;
397 }
398 tr_exit ( kid2 );
399 break;
400 default:
401 tr_entry ( kid2 );
402 len_error
403 ( 1,
404 /*IMSC*/"internal error: do_reset_seq: bad node name: ts"
405 , syntostr ( nameofnode ( kid2 ) )
406 );
407 tr_exit ( kid2 );
408 break;
409 }
410 while ( start < the->reset_cnt )
411 { the->sequences [ start ] . bits = kid;
412 the->sequences [ start ] . fb_kid = fb;
413 ++start;
414 }
415 tr_exit ( kid );
416 tr_exit ( T );
417 *pre_length_addr = pre_len;
418 *post_length_addr = post_len;
419 } /* do_reset_seq */
420
421 /* end of reset sequence handling code */
422
423 /* start of timing specification handling code */
424
425 void
426 check_state ( T )
427 tree T;
428 /* checks the state identifier passed in T */
429 { symbol text;
430 tr_entry ( T );
431 text = textofnode ( T );
432 if ( !compare ( text, C_low ) )
433 set_text ( T, C_low );
434 else if ( !compare ( text, C_high ) )
435 set_text ( T, C_high );
436 else if ( !compare ( text, C_float ) )
437 set_text ( T, C_float );
438 else if ( !compare ( text, C_valid ) )
439 set_text ( T, C_valid );
440 else if ( !compare ( text, C_asy ) )
441 set_text ( T, C_asy );
442 else
443 error
444 ( T
445 /*IMSC*/"illegal signal state: ts (must be low, high, float, valid, or asy)"
446 , syntostr ( text )
447 );
448 tr_exit ( T );
449 } /* check_state */
450
451 static unsigned long
452 do_spec ( T )
453 tree T;
454 /* process a single time specification, returns time in picoseconds */
455 { ushort kids;
456 symbol text;
457 double val;
458 tree kid;
459 tr_entry ( T );
460 kids = kidcount ( T );
461 if ( kids == 2 )
462 { kid = subtree ( 1, T );
463 tr_entry ( kid );
464 text = textofnode ( kid );
465 if ( !compare ( text, C_min ) )
466 set_text ( kid, C_min );
467 else if ( !compare ( text, C_typ ) )
468 set_text ( kid, C_typ );
469 else if ( !compare ( text, C_max ) )
470 set_text ( kid, C_max );
471 else
472 error
473 ( kid
474 /*IMSC*/"illegal time specifier: ts (must be min, typ, or max)"
475 , syntostr ( text )
476 );
477 tr_exit ( kid );
478 }
479 val = check_sum ( subtree ( kids, T ) );
480

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE 5/23/89	PAGE # 5/81
LINE #	SOURCE TEXT			
481	/* some kind of range check? */			
482	tr_exit (T)			
483	return val * 1000 + 0.5 ;			
484	/* do_spec */			
485	void			
487	process_timing (T , min_addr , typ_addr , max_addr)			
488	tree T ;			
489	unsigned long			
490	*min_addr ,			
491	*typ_addr ,			
492	*max_addr ;			
493	/* process a timing specification triple, returning min/typ/max */			
494	{ ushort			
495	kids ;			
496	bas1 ;			
497	bas2 ;			
498	bas3 ;			
499	tree			
500	fkid ,			
501	lkid ,			
502	kid ,			
503	unsigned long			
504	spec1 ,			
505	spec2 ,			
506	spec3 ;			
507	tr_entry (T) ,			
508	*min_addr =			
509	*typ_addr =			
510	*max_addr =			
511	-1 ;			
512	kids = kidcount (T) ;			
513	fkid = subtree (1 , T) ;			
514	lkid = subtree (kids , T) ;			
515	switch (kids)			
516	{ case 1 :			
517	spec1 = do_spec (fkid) ;			
518	bas1 = kidcount (fkid) - 2 ;			
519	if (bas1			
520	switch (textofnode (subtree (1 , fkid)))			
521	{ case C_min :			
522	{ case C_typ :			
523	*min_addr =			
524	*typ_addr =			
525	*max_addr =			
526	spec1 ;			
527	break ;			
528	case C_max :			
529	*min_addr = 0 ;			
530	*typ_addr =			
531	*max_addr =			
532	spec1 ;			
533	break ;			
534	default :			
535	break ;			
536	}			
537	else			
538	{ *min_addr = 0 ;			
539	*typ_addr =			
540	*max_addr =			
541	spec1 ;			
542	}			
543	break ;			
544	case 2 :			
545	spec1 = do_spec (fkid) ;			
546	spec2 = do_spec (lkid) ;			
547	bas1 = kidcount (fkid) - 2 ;			
548	bas2 = kidcount (lkid) - 2 ;			
549	if (bas1 && bas2)			
550	switch (textofnode (subtree (1 , fkid)))			
551	{ case C_min :			
552	*min_addr = spec1 ;			
553	switch (textofnode (subtree (1 , lkid)))			
554	{ case C_min :			
555	error			
556	(subtree (1 , lkid) ,			
557	/*MSG*/"minimum delay respecified"			
558);			
559	break ;			
560	case C_typ :			
561	*typ_addr =			
562	*max_addr =			
563	spec2 ;			
564	break ;			
565	case C_max :			
566	*typ_addr = (spec1 + spec2) / 2 ;			
567	*max_addr = spec2 ;			
568	break ;			
569	default :			
570	break ;			
571	}			
572	break ;			
573	case C_typ :			
574	*typ_addr = spec1 ;			
575	switch (textofnode (subtree (1 , lkid)))			
576	{ case C_min :			
577	*min_addr = spec2 ;			
578	*max_addr = spec1 ;			
579	break ;			
580	case C_typ :			
581	error			
582	(subtree (1 , lkid) ,			
583	/*MSG*/"typical delay respecified"			
584);			
585	break ;			
586	case C_max :			
587	*min_addr = spec1 ;			
588	*max_addr = spec2 ;			
589	break ;			
590	default :			
591	break ;			
592	}			
593	break ;			
594	case C_max :			
595	switch (textofnode (subtree (1 , lkid)))			
596	{ case C_min :			
597	*min_addr = spec2 ;			
598	*typ_addr = (spec1 + spec2) / 2 ;			
599	break ;			
600	case C_typ :			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

6/82

TIME

4:42:27 pm

LINE # SOURCE TEXT

```
601      *min_addr =
602      *typ_addr =
603      spec1 ;
604      break ;
605      case C_max :
606      error
607      ( subtree ( 1 , lkid ) ,
608      /*IMSC*/"maximum delay respecified"
609      ) ;
610      break ;
611      default :
612      break ;
613      }
614      break ;
615      default :
616      break ;
617      }
618      else if ( has1 )
619      error
620      ( subtree ( 1 , fkid ) ,
621      /*IMSC*/"ambiguous delay specification"
622      ) ;
623      else if ( has2 )
624      error
625      ( subtree ( 1 , lkid ) ,
626      /*IMSC*/"ambiguous delay specification"
627      ) ;
628      else
629      { *min_addr = spec1 ;
630      *typ_addr = ( spec1 + spec2 ) / 2 ;
631      *max_addr = spec2 ;
632      }
633      break ;
634      case 3 :
635      kid = subtree ( 2 , T ) ;
636      spec1 = do_spec ( fkid ) ;
637      spec2 = do_spec ( lkid ) ;
638      spec3 = do_spec ( lkid ) ;
639      has1 = kidcount ( fkid ) == 2 ;
640      has2 = kidcount ( lkid ) == 2 ;
641      has3 = kidcount ( lkid ) == 2 ;
642      if ( has1 && has2 && has3 )
643      switch ( textofnode ( subtree ( 1 , fkid ) ) )
644      { case C_min :
645      { *min_addr = spec1 ;
646      switch ( textofnode ( subtree ( 1 , lkid ) ) )
647      { case C_min :
648      error
649      ( subtree ( 1 , lkid ) ,
650      /*IMSC*/"minimum delay respecified"
651      ) ;
652      switch ( textofnode ( subtree ( 1 , lkid ) ) )
653      { case C_min :
654      error
655      ( subtree ( 1 , lkid ) ,
656      /*IMSC*/"minimum delay respecified"
657      ) ;
658      break ;
659      case C_typ :
660      *typ_addr = spec3 ;
661      break ;
662      case C_max :
663      *max_addr = spec3 ;
664      break ;
665      default :
666      break ;
667      }
668      break ;
669      case C_typ :
670      *typ_addr = spec2 ;
671      switch ( textofnode ( subtree ( 1 , lkid ) ) )
672      { case C_min :
673      error
674      ( subtree ( 1 , lkid ) ,
675      /*IMSC*/"minimum delay respecified"
676      ) ;
677      break ;
678      case C_typ :
679      error
680      ( subtree ( 1 , lkid ) ,
681      /*IMSC*/"typical delay respecified"
682      ) ;
683      break ;
684      case C_max :
685      *max_addr = spec3 ;
686      break ;
687      default :
688      break ;
689      }
690      break ;
691      case C_max :
692      *max_addr = spec2 ;
693      switch ( textofnode ( subtree ( 1 , lkid ) ) )
694      { case C_min :
695      error
696      ( subtree ( 1 , lkid ) ,
697      /*IMSC*/"minimum delay respecified"
698      ) ;
699      break ;
700      case C_typ :
701      *typ_addr = spec3 ;
702      break ;
703      case C_max :
704      error
705      ( subtree ( 1 , lkid ) ,
706      /*IMSC*/"maximum delay respecified"
707      ) ;
708      break ;
709      default :
710      break ;
711      }
712      break ;
713      default :
714      switch ( textofnode ( subtree ( 1 , lkid ) ) )
715      { case C_min :
716      error
717      ( subtree ( 1 , lkid ) ,
718      /*IMSC*/"minimum delay respecified"
719      ) ;
720      break ;
```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE 5/23/89	PAGE # 7/83
LINE #	SOURCE TEXT			
721	case C_typ :			
722	*typ_addr = spec1 ;			
723	break ;			
724	case C_max :			
725	*max_addr = spec1 ;			
726	break ;			
727	default :			
728	break ;			
729	}			
730	break ;			
731	}			
732	break ;			
733	case C_typ :			
734	*typ_addr = spec1 ;			
735	switch (textofnode (subtree (1 , kid)))			
736	{ case C_min :			
737	*min_addr = spec2 ;			
738	switch (textofnode (subtree (1 , lkid)))			
739	{ case C_min :			
740	error			
741	(subtree (1 , kid) ,			
742	/*LMSC*/"minimum delay respecified"			
743	{			
744	break ;			
745	case C_typ :			
746	error			
747	(subtree (1 , kid) ,			
748	/*LMSC*/"typical delay respecified"			
749	{			
750	break ;			
751	case C_max :			
752	*max_addr = spec3 ;			
753	break ;			
754	default :			
755	break ;			
756	}			
757	break ;			
758	case C_typ :			
759	error			
760	(subtree (1 , kid) ,			
761	/*LMSC*/"typical delay respecified"			
762	{			
763	switch (textofnode (subtree (1 , lkid)))			
764	{ case C_min :			
765	*min_addr = spec3 ;			
766	break ;			
767	case C_typ :			
768	error			
769	(subtree (1 , lkid) ,			
770	/*LMSC*/"typical delay respecified"			
771	{			
772	break ;			
773	case C_max :			
774	*max_addr = spec3 ;			
775	break ;			
776	default :			
777	break ;			
778	}			
779	break ;			
780	case C_max :			
781	*max_addr = spec2 ;			
782	switch (textofnode (subtree (1 , lkid)))			
783	{ case C_min :			
784	*min_addr = spec3 ;			
785	break ;			
786	case C_typ :			
787	error			
788	(subtree (1 , lkid) ,			
789	/*LMSC*/"typical delay respecified"			
790	{			
791	break ;			
792	case C_max :			
793	error			
794	(subtree (1 , lkid) ,			
795	/*LMSC*/"maximum delay respecified"			
796	{			
797	break ;			
798	default :			
799	break ;			
800	}			
801	break ;			
802	default :			
803	switch (textofnode (subtree (1 , lkid)))			
804	{ case C_min :			
805	*min_addr = spec3 ;			
806	break ;			
807	case C_typ :			
808	error			
809	(subtree (1 , lkid) ,			
810	/*LMSC*/"typical delay respecified"			
811	{			
812	break ;			
813	case C_max :			
814	*max_addr = spec3 ;			
815	break ;			
816	default :			
817	break ;			
818	}			
819	break ;			
820	}			
821	break ;			
822	case C_max :			
823	*max_addr = spec1 ;			
824	switch (textofnode (subtree (1 , kid)))			
825	{ case C_min :			
826	*min_addr = spec2 ;			
827	switch (textofnode (subtree (1 , lkid)))			
828	{ case C_min :			
829	error			
830	(subtree (1 , kid) ,			
831	/*LMSC*/"minimum delay respecified"			
832	{			
833	break ;			
834	case C_typ :			
835	*typ_addr = spec3 ;			
836	break ;			
837	case C_max :			
838	error			
839	(subtree (1 , kid) ,			
840	/*LMSC*/"maximum delay respecified"			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE : 5/23/89 TIME 4:42:27 pm	PAGE # 8/84
LINE #	SOURCE TEXT			
841	}			
842	break ;			
843	default :			
844	break ;			
845	}			
846	break ;			
847	case C_typ :			
848	"typ_addr = spec2 ;			
849	switch (textofnode (subtree (1 , lkid)))			
850	{ case C_min :			
851	"min_addr = spec3 ;			
852	break ;			
853	case C_typ :			
854	cerror			
855	(subtree (1 , lkid) ,			
856	/*IMSC*/"typical delay respecified"			
857	{			
858	break ;			
859	case C_max :			
860	cerror			
861	(subtree (1 , lkid) ,			
862	/*IMSC*/"maximum delay respecified"			
863	{			
864	break ;			
865	default :			
866	break ;			
867	}			
868	break ;			
869	case C_max :			
870	cerror			
871	(subtree (1 , lkid) ,			
872	/*IMSC*/"maximum delay respecified"			
873	{			
874	switch (textofnode (subtree (1 , lkid)))			
875	{ case C_min :			
876	"min_addr = spec3 ;			
877	break ;			
878	case C_typ :			
879	"typ_addr = spec3 ;			
880	break ;			
881	case C_max :			
882	cerror			
883	(subtree (1 , lkid) ,			
884	/*IMSC*/"maximum delay respecified"			
885	{			
886	break ;			
887	default :			
888	break ;			
889	}			
890	break ;			
891	default :			
892	switch (textofnode (subtree (1 , lkid)))			
893	{ case C_min :			
894	"min_addr = spec3 ;			
895	break ;			
896	case C_typ :			
897	"typ_addr = spec3 ;			
898	break ;			
899	case C_max :			
900	cerror			
901	(subtree (1 , lkid) ,			
902	/*IMSC*/"maximum delay respecified"			
903	{			
904	break ;			
905	default :			
906	break ;			
907	}			
908	break ;			
909	}			
910	break ;			
911				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_4.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
9/85

LINE #	SOURCE TEXT
961	error
962	(subtree (1 , lkid) ,
963	/*IMSC*/"maximum delay respesified"
964);
965	break ;
966	default :
967	break ;
968	break ;
969	break ;
970	default :
971	switch (textofnode (subtree (1 , lkid)))
972	{ case C_min :
973	min_addr = spec3 ;
974	break ;
975	case C_typ :
976	typ_addr = spec3 ;
977	break ;
978	case C_max :
979	max_addr = spec3 ;
980	break ;
981	default :
982	break ;
983	break ;
984	break ;
985	break ;
986	break ;
987	break ;
988	else if (has2 && has3)
989	switch (textofnode (subtree (1 , lkid)))
990	{ case C_min :
991	min_addr = spec1 ;
992	switch (textofnode (subtree (1 , lkid)))
993	{ case C_min :
994	error
995	(subtree (1 , lkid) ,
996	/*IMSC*/"minimum delay respesified"
997);
998	break ;
999	case C_typ :
1000	typ_addr = spec2 ;
1001	max_addr = spec3 ;
1002	break ;
1003	case C_max :
1004	max_addr = spec2 ;
1005	typ_addr = spec3 ;
1006	break ;
1007	default :
1008	break ;
1009	break ;
1010	break ;
1011	case C_typ :
1012	typ_addr = spec1 ;
1013	switch (textofnode (subtree (1 , lkid)))
1014	{ case C_min :
1015	min_addr = spec2 ;
1016	max_addr = spec3 ;
1017	break ;
1018	break ;
1019	case C_typ :
1020	error
1021	(subtree (1 , lkid) ,
1022	/*IMSC*/"typical delay respesified"
1023);
1024	break ;
1025	case C_max :
1026	max_addr = spec2 ;
1027	min_addr = spec3 ;
1028	break ;
1029	default :
1030	break ;
1031	break ;
1032	case C_max :
1033	max_addr = spec1 ;
1034	switch (textofnode (subtree (1 , lkid)))
1035	{ case C_min :
1036	min_addr = spec2 ;
1037	typ_addr = spec3 ;
1038	break ;
1039	case C_typ :
1040	typ_addr = spec2 ;
1041	min_addr = spec3 ;
1042	break ;
1043	case C_max :
1044	error
1045	(subtree (1 , lkid) ,
1046	/*IMSC*/"maximum delay respesified"
1047);
1048	break ;
1049	default :
1050	break ;
1051	break ;
1052	break ;
1053	break ;
1054	default :
1055	switch (textofnode (subtree (1 , lkid)))
1056	{ case C_min :
1057	min_addr = spec2 ;
1058	break ;
1059	case C_typ :
1060	typ_addr = spec2 ;
1061	break ;
1062	case C_max :
1063	max_addr = spec2 ;
1064	break ;
1065	default :
1066	break ;
1067	break ;
1068	break ;
1069	else if (has2 && has3)
1070	switch (textofnode (subtree (1 , lkid)))
1071	{ case C_min :
1072	min_addr = spec2 ;
1073	switch (textofnode (subtree (1 , lkid)))
1074	{ case C_min :
1075	error
1076	(subtree (1 , lkid) ,
1077	/*IMSC*/"minimum delay respesified"
1078);
1079	break ;
1080	case C_typ :

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_4.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
10/86

```

LINE # SOURCE TEXT
1081      *typ_addr = spec3 ;
1082      *max_addr = spec1 ;
1083      break ;
1084      case C_max :
1085      *max_addr = spec3 ;
1086      *typ_addr = spec1 ;
1087      break ;
1088      default :
1089      break ;
1090
1091      break ;
1092      case C_typ :
1093      *typ_addr = spec2 ;
1094      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1095      { case C_min :
1096      *min_addr = spec3 ;
1097      *max_addr = spec1 ;
1098      break ;
1099      case C_typ :
1100      *typ_addr = spec3 ;
1101      *max_addr = spec1 ;
1102      break ;
1103      /*IMSC*/"typical delay respecified"
1104      {
1105      break ;
1106      case C_max :
1107      *max_addr = spec3 ;
1108      *min_addr = spec1 ;
1109      break ;
1110      default :
1111      break ;
1112      }
1113      break ;
1114      case C_max :
1115      *max_addr = spec2 ;
1116      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1117      { case C_min :
1118      *min_addr = spec3 ;
1119      *typ_addr = spec1 ;
1120      break ;
1121      case C_typ :
1122      *typ_addr = spec3 ;
1123      *min_addr = spec1 ;
1124      break ;
1125      case C_max :
1126      *max_addr = spec3 ;
1127      *typ_addr = spec1 ;
1128      break ;
1129      /*IMSC*/"maximum delay respecified"
1130      {
1131      break ;
1132      default :
1133      break ;
1134      }
1135      break ;
1136      default :
1137      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1138      { case C_min :
1139      *min_addr = spec3 ;
1140      break ;
1141      case C_typ :
1142      *typ_addr = spec3 ;
1143      break ;
1144      case C_max :
1145      *max_addr = spec3 ;
1146      break ;
1147      default :
1148      break ;
1149      }
1150      break ;
1151      else if ( hval <= hval )
1152      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1153      { case C_min :
1154      *min_addr = spec1 ;
1155      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1156      { case C_min :
1157      *min_addr = spec1 ;
1158      *typ_addr = spec3 ;
1159      break ;
1160      /*IMSC*/"minimum delay respecified"
1161      {
1162      break ;
1163      case C_typ :
1164      *typ_addr = spec3 ;
1165      *max_addr = spec2 ;
1166      break ;
1167      case C_max :
1168      *max_addr = spec3 ;
1169      *typ_addr = spec2 ;
1170      break ;
1171      default :
1172      break ;
1173      }
1174      break ;
1175      case C_typ :
1176      *typ_addr = spec1 ;
1177      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1178      { case C_min :
1179      *min_addr = spec3 ;
1180      *max_addr = spec2 ;
1181      break ;
1182      case C_typ :
1183      *typ_addr = spec3 ;
1184      *max_addr = spec2 ;
1185      break ;
1186      /*IMSC*/"typical delay respecified"
1187      {
1188      break ;
1189      case C_max :
1190      *max_addr = spec3 ;
1191      *min_addr = spec2 ;
1192      break ;
1193      default :
1194      break ;
1195      }
1196      break ;
1197      case C_max :
1198      *max_addr = spec1 ;
1199      switch ( tnodefnode ( subtree ( 1 , lkid ) ) )
1200      { case C_min :
1201      *min_addr = spec3 ;
1202      *typ_addr = spec2 ;
1203      break ;

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE 5/23/89	PAGE # 11/87
LINE #	SOURCE TEXT			
1201	case C_typ :			
1202	*typ_addr = spec3 ;			
1203	*min_addr = spec2 ;			
1204	break ;			
1205	/* C_max :			
1206	error			
1207	(subtree (1 , lkid) ,			
1208	/*IMSG*/"maximum delay respecified"			
1209) ;			
1210	break ;			
1211	default :			
1212	break ;			
1213	break ;			
1214	break ;			
1215	default :			
1216	switch (textofnode (subtree (1 , lkid)))			
1217	{ case C_min :			
1218	*min_addr = spec3 ;			
1219	break ;			
1220	case C_typ :			
1221	*typ_addr = spec3 ;			
1222	break ;			
1223	case C_max :			
1224	*max_addr = spec3 ;			
1225	break ;			
1226	default :			
1227	break ;			
1228	}			
1229	break ;			
1230	}			
1231	else if (haal)			
1232	error			
1233	(subtree (1 , fkid) ,			
1234	/*IMSG*/"ambiguous delay specification"			
1235) ;			
1236	else if (haal)			
1237	error			
1238	(subtree (1 , lkid) ,			
1239	/*IMSG*/"ambiguous delay specification"			
1240) ;			
1241	else if (haal)			
1242	error			
1243	(subtree (1 , lkid) ,			
1244	/*IMSG*/"ambiguous delay specification"			
1245) ;			
1246	else			
1247	{ *min_addr = spec1 ;			
1248	*typ_addr = spec2 ;			
1249	*max_addr = spec3 ;			
1250	}			
1251	break ;			
1252	default :			
1253	lm_error			
1254	(1 ,			
1255	/*IMSG*/"internal error: process timing: bad kidcount"			
1256) ;			
1257	break ;			
1258	}			
1259	if (*min_addr != -1)			
1260	{ if (*typ_addr != -1 && *min_addr > *typ_addr)			
1261	error			
1262	(T ,			
1263	/*IMSG*/"minimum time more than typical time"			
1264) ;			
1265	if (*max_addr != -1 && *min_addr > *max_addr)			
1266	error			
1267	(T ,			
1268	/*IMSG*/"minimum time more than maximum time"			
1269) ;			
1270	}			
1271	if (*typ_addr != -1 && *max_addr != -1 && *typ_addr > *max_addr)			
1272	error			
1273	(T ,			
1274	/*IMSG*/"typical time more than maximum time"			
1275) ;			
1276	tr_exit (T) ;			
1277	} /* process timing */			
1278	}			
1279	static void			
1280	add_delay			
1281	(pin ,			
1282	maj_st ,			
1283	min_st ,			
1284	pin_name ,			
1285	timing			
1286)			
1287	{ ushort pin ;			
1288	symbol			
1289	maj_st ,			
1290	min_st ,			
1291	pin_name ,			
1292	tree timing ;			
1293	/* adds a timing structure to the list for the passed pin with passed info */			
1294	{ struct timing			
1295	*list ,			
1296	*delay ;			
1297	delay = allo_timing () ;			
1298	delay->major_state = maj_st ;			
1299	delay->minor_state = min_st ;			
1300	delay->minor_pin_name = pin_name ;			
1301	delay->next = NULL ;			
1302	process timing			
1303	{ timing ,			
1304	& delay->minimum ,			
1305	& delay->typical ,			
1306	& delay->maximum			
1307	}			
1308	if ((list = the->pins [pin] . timing_list) == NULL)			
1309	the->pins [pin] . timing_list = delay ;			
1310	else			
1311	{ do			
1312	{ if			
1313	{ list->major_state == maj_st &&			
1314	list->minor_state == min_st &&			
1315	list->minor_pin_name == pin_name			
1316	}			
1317	error			
1318	(subtree (1 , subtree (1 , timing)) ,			
1319	/*IMSG*/"duplicate delay specification"			
1320) ;			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

12/88

TIME 4:42:27 pm

```

1321 while ( list -> next != NULL && ( list = list -> next , 1 ) ) ,
1322 list -> next = delay ;
1323 }
1324 } /* add_delay */
1325
1326 static void
1327 build_delays ( from , state , text , timing )
1328 tree
1329 from ,
1330 state ,
1331 timing ,
1332 symbol text ;
1333 /* builds the list of delay specs passed via add_delay() */
1334 { ushort
1335 i ,
1336 k ,
1337 kids ,
1338 pin ,
1339 sum1 ,
1340 sum2 ,
1341 symbol
1342 maj_st ,
1343 min_st ,
1344 text2 ,
1345 tree
1346 kid2 ,
1347 kid ,
1348 pin = pin_name_of ( text ) ;
1349 maj_st = textofnode ( state ) ;
1350 if ( from == NULLTEXT )
1351 { add_delay
1352 ( pin ,
1353 maj_st ,
1354 NULLSYM ,
1355 NULLSYM ,
1356 timing
1357 ) ,
1358 return ;
1359 }
1360 min_st = textofnode ( subtree ( 1 , from ) ) ;
1361 kids = kidcount ( from ) ;
1362 for ( k = 2 , k <= kids , ++k )
1363 { kid = subtree ( k , from ) ;
1364 switch ( nameofnode ( kid ) )
1365 { case P_ID :
1366 case P_NUM :
1367 add_delay
1368 ( pin ,
1369 maj_st ,
1370 min_st ,
1371 textofnode ( kid ) ,
1372 timing
1373 ) ,
1374 break ;
1375 case P_STR :
1376 /* what about generators in strings? */
1377 buf_string ( symtostr ( textofnode ( kid ) ) ) ,
1378 text2 = strtosym ( buf ) ,
1379 add_delay
1380 ( pin ,
1381 maj_st ,
1382 min_st ,
1383 text2 ,
1384 timing
1385 ) ,
1386 break ;
1387 case N_PIN_NAME :
1388 kid2 = subtree ( 1 , kid ) ;
1389 text2 = textofnode ( kid2 ) ;
1390 kid2 = subtree ( 2 , kid ) ;
1391 sum1 =
1392 sum2 =
1393 dec_ratio ( kid2 ) ;
1394 if ( kidcount ( kid ) == 3 )
1395 { kid2 = subtree ( 3 , kid ) ;
1396 sum2 = dec_ratio ( kid2 ) ;
1397 }
1398 if ( sum1 > sum2 )
1399 for ( i = sum1 , --i )
1400 { add_delay
1401 ( pin ,
1402 maj_st ,
1403 min_st ,
1404 gen_name ( text2 , i ) ,
1405 timing
1406 ) ,
1407 if ( i == sum2 )
1408 break ;
1409 }
1410 else
1411 for ( i = sum1 , ++i )
1412 { add_delay
1413 ( pin ,
1414 maj_st ,
1415 min_st ,
1416 gen_name ( text2 , i ) ,
1417 timing
1418 ) ,
1419 if ( i == sum2 )
1420 break ;
1421 }
1422 break ;
1423 default :
1424 in_error
1425 ( 1 ,
1426 /*LMSC*/"internal error: build_delays: unknown node name: %s"
1427 , symtostr ( nameofnode ( kid ) ) )
1428 }
1429 }
1430 } /* build_delays */
1431
1432 static void
1433 do_outputs ( from , state , T , text , timing )
1434 tree
1435 from ,
1436 state ,
1437 T ,
1438 timing ,
1439 symbol text ;
1440

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE 5/23/89	PAGE # 13/89
LINE #		SOURCE TEXT		
1441		/* check that text is an output or I/O, then call build_delays() */		
1442		{ tree look ;		
1443		look = g_lookup (text) ;		
1444		if (look == NULLTREE)		
1445		{ error		
1446		{ T		
1447		/*MSG*/"no such device signal name: %s"		
1448		, systr (text)		
1449		}		
1450		return ;		
1451		}		
1452		else		
1453		switch (nameofnode (look))		
1454		{ case N_IN_PINS :		
1455		case N_POWER_PINS :		
1456		case N_GROUND_PINS :		
1457		case N_NC_PINS :		
1458		error		
1459		{ T		
1460		/*MSG*/"device signal name %s is not an out_pin or io_pin"		
1461		, systr (text)		
1462		}		
1463		return ;		
1464		case N_OUT_PINS :		
1465		case N_IO_PINS :		
1466		break ;		
1467		default :		
1468		error		
1469		{ T		
1470		/*MSG*/"%s is not a device signal name"		
1471		, systr (text)		
1472		}		
1473		return ;		
1474		}		
1475		build_delays		
1476		{ from ,		
1477		state ,		
1478		text ,		
1479		timing		
1480		}		
1481		} /* do_outputs */		
1482		}		
1483		is_store_or_eval (text)		
1484		symbol text ;		
1485		/* returns 1 iff the passed pin name is a store or eval pin */		
1486		{ ushort		
1487		{		
1488		index ;		
1489		index = pin_name_of (text) ;		
1490		for (i = 0 ; i < the -> pins [index] . num_attr ; ++i)		
1491		if (the -> pins [index] . attr [i] . use_flag == ID_ATTR)		
1492		{ text = the -> pins [index] . attr [i] . attr . id ;		
1493		if		
1494		{ text == C_eval		
1495		text == C_store		
1496		text == C_edge_rise		
1497		text == C_edge_fall		
1498		}		
1499		return 1 ;		
1500		}		
1501		return 0 ;		
1502		} /* is_store_or_eval */		
1503		static ushort		
1504		do_stores (T , text)		
1505		{ tree T ;		
1506		symbol text ;		
1507		/* makes sure the passed pin name is an in or I/O, and a store or eval */		
1508		/* returns 1 iff the passed pin name is okay */		
1509		{ tree look ;		
1510		look = g_lookup (text) ;		
1511		if (look == NULLTREE)		
1512		{ error		
1513		{ T		
1514		/*MSG*/"no such device signal name: %s"		
1515		, systr (text)		
1516		}		
1517		return 0 ;		
1518		}		
1519		else		
1520		switch (nameofnode (look))		
1521		{ case N_IN_PINS :		
1522		case N_IO_PINS :		
1523		if (! is_store_or_eval (text))		
1524		{ error		
1525		{ T		
1526		/*MSG*/"device signal name %s is not a store or eval pin"		
1527		, systr (text)		
1528		}		
1529		return 0 ;		
1530		}		
1531		break ;		
1532		case N_OUT_PINS :		
1533		case N_POWER_PINS :		
1534		case N_GROUND_PINS :		
1535		case N_NC_PINS :		
1536		error		
1537		{ T		
1538		/*MSG*/"device signal name %s is not an in_pin or io_pin"		
1539		, systr (text)		
1540		}		
1541		return 0 ;		
1542		default :		
1543		error		
1544		{ T		
1545		/*MSG*/"%s is not a device signal name"		
1546		, systr (text)		
1547		}		
1548		return 0 ;		
1549		}		
1550		return 1 ;		
1551		} /* do_stores */		
1552		static void		
1553		check_outputs (from , state , T , timing)		
1554		{ tree		
1555		from ,		
1556		state ,		
1557		T ;		
1558		}		
1559				
1560				

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE

5/23/89

PAGE #

TIME

4:42:27 pm

14/90

```

1561 timing ;
1562 /* makes sure the passed pins in T are all outputs */
1563 /* continues processing via do_outputs() */
1564 { ushort
1565     i ;
1566     sum1 /*= 0*/ ;
1567     sum2 /*= 0*/ ;
1568     symbol text ;
1569     tree kid ;
1570     ( void ) do_pin_name ( T ) ;
1571     switch ( nameofnode ( T ) )
1572     { case P_ID :
1573         case P_NUM :
1574             do_outputs
1575             ( from ,
1576               state ,
1577               T ,
1578               textofnode ( T ) ,
1579               timing
1580             ) ;
1581             break ;
1582         case P_STR :
1583             /* what about generators in strings? */
1584             buf_string ( systostr ( textofnode ( T ) ) ) ;
1585             text = strtoysa ( buf ) ;
1586             do_outputs
1587             ( from ,
1588               state ,
1589               T ,
1590               text ,
1591               timing
1592             ) ;
1593             break ;
1594         case N_PIN_NAME :
1595             kid = subtree ( 1 , T ) ;
1596             text = textofnode ( kid ) ;
1597             kid = subtree ( 2 , T ) ;
1598             sum1 =
1599             sum2 =
1600             dec_ration ( kid ) ;
1601             if ( kidcount ( T ) == 3 )
1602             { kid = subtree ( 3 , T ) ;
1603               sum1 = dec_ration ( kid ) ;
1604             }
1605             if ( sum1 > sum2 )
1606             for ( i = sum1 ; --i )
1607             { do_outputs
1608               ( from ,
1609                 state ,
1610                 T ,
1611                 gen_name ( text , i ) ,
1612                 timing
1613               ) ;
1614               if ( i == sum2 )
1615                 break ;
1616             }
1617             else
1618             for ( i = sum1 ; ++i )
1619             { do_outputs
1620               ( from ,
1621                 state ,
1622                 T ,
1623                 gen_name ( text , i ) ,
1624                 timing
1625               ) ;
1626               if ( i == sum2 )
1627                 break ;
1628             }
1629             break ;
1630         default :
1631             error
1632             ( T ,
1633             /*MSG*/"internal error: check_outputs: unknown node name: %s"
1634             ) ;
1635             systostr ( nameofnode ( T ) ) ;
1636             break ;
1637     }
1638 } /* check_outputs */
1639
1640 static void
1641 check_stores ( T )
1642 { tree T ;
1643   /* makes sure the passed pins in T are all stores */
1644   /* continues processing via do_stores() */
1645   { ushort
1646       i ;
1647       sum1 /*= 0*/ ;
1648       sum2 /*= 0*/ ;
1649       symbol text ;
1650       tree kid ;
1651       ( void ) do_pin_name ( T ) ;
1652       switch ( nameofnode ( T ) )
1653       { case P_ID :
1654           case P_NUM :
1655               ( void ) do_stores ( T , textofnode ( T ) ) ;
1656               break ;
1657           case P_STR :
1658               /* what about generators in strings? */
1659               buf_string ( systostr ( textofnode ( T ) ) ) ;
1660               text = strtoysa ( buf ) ;
1661               ( void ) do_stores ( T , text ) ;
1662               break ;
1663           case N_PIN_NAME :
1664               kid = subtree ( 1 , T ) ;
1665               text = textofnode ( kid ) ;
1666               kid = subtree ( 2 , T ) ;
1667               sum1 =
1668               sum2 =
1669               dec_ration ( kid ) ;
1670               if ( kidcount ( T ) == 3 )
1671               { kid = subtree ( 3 , T ) ;
1672                 sum2 = dec_ration ( kid ) ;
1673               }
1674               if ( sum1 > sum2 )
1675               for ( i = sum1 ; --i )
1676               { ( void ) do_stores ( T , gen_name ( text , i ) ) ;
1677                 if ( i == sum2 )
1678                   break ;
1679               }
1680               else

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_4.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
15/91

```

LINE # SOURCE TEXT
1681 for ( i = sum1 ; ++i )
1682 { ( void ) do_stores ( T , gen_name ( text , i ) ) ;
1683 if ( i == sum2 )
1684 break ;
1685 }
1686 break ;
1687 default :
1688 error
1689 { T ,
1690 /*LMSG*/"internal error: check_stores: unknown node name: %s"
1691 , syntestr ( nameofnode ( T ) )
1692 } ;
1693 break ;
1694 }
1695 } /* check_stores */
1696
1697 static void
1698 do_to_pinstate ( from , T , timing )
1699 {
1700 from ,
1701 T ,
1702 timing ,
1703 /* checks the to_pinstate and its pins via check_outputs() */
1704 { ushort
1705 k ,
1706 kids ,
1707 tr_entry ( T ) ,
1708 check_state ( subtree ( 1 , T ) ) ,
1709 kids = kidcount ( T ) ,
1710 for ( k = 2 ; k <= kids ; ++k )
1711 check_outputs (
1712 from ,
1713 subtree ( 1 , T ) ,
1714 subtree ( k , T ) ,
1715 timing
1716 ) ;
1717 tr_exit ( T ) ;
1718 } /* do_to_pinstate */
1719
1720 static void
1721 do_from_pinstate ( T )
1722 {
1723 tree T ,
1724 /* checks the from_pinstate and its pins via check_stores() */
1725 { ushort
1726 k ,
1727 kids ,
1728 symbol text ,
1729 tr_entry ( T ) ,
1730 check_state ( subtree ( 1 , T ) ) ,
1731 text = textofnode ( subtree ( 1 , T ) ) ,
1732 if ( text == C_float || text == C_any )
1733 error
1734 { subtree ( 1 , T ) ,
1735 /*LMSG*/"illegal delay from-pin state: %s (must not be float or any)"
1736 , syntestr ( text )
1737 } ;
1738 kids = kidcount ( T ) ,
1739 for ( k = 2 ; k <= kids ; ++k )
1740 check_stores ( subtree ( k , T ) ) ,
1741 tr_exit ( T ) ;
1742 } /* do_from_pinstate */
1743
1744 static void
1745 do_to_delay ( from , T )
1746 {
1747 from ,
1748 T ,
1749 /* processes the to_delay construct via do_to_pinstate() */
1750 { ushort kids ,
1751 tr_entry ( T ) ,
1752 kids = kidcount ( T ) ,
1753 do_to_pinstate
1754 { from ,
1755 subtree ( 1 , T ) ,
1756 subtree ( kids , T )
1757 } ;
1758 tr_exit ( T ) ;
1759 } /* do_to_delay */
1760
1761 static void
1762 do_from_delay ( T )
1763 {
1764 tree T ,
1765 /* processes the from_delay construct via do_to_delay() */
1766 { ushort
1767 k ,
1768 kids ,
1769 tr_entry ( T ) ,
1770 do_from_pinstate ( subtree ( 1 , T ) ) ,
1771 kids = kidcount ( T ) ,
1772 for ( k = 2 ; k <= kids ; ++k )
1773 do_to_delay ( subtree ( 1 , T ) , subtree ( k , T ) ) ,
1774 tr_exit ( T ) ;
1775 } /* do_from_delay */
1776
1777 /* end of timing specification handling code */
1778
1779 /* start of trace delay handling code */
1780
1781 static void
1782 do_trace_delays ( T )
1783 {
1784 tree T ,
1785 { ushort
1786 k ,
1787 kids ,
1788 sum ,
1789 td_seen = 0 ,
1790 tree
1791 { kid ,
1792 attr ,
1793 delay ,
1794 symbol name ,
1795 double val ,
1796 tr_entry ( T ) ,
1797 kids = kidcount ( T ) ,
1798 for ( k = 3 ; k <= kids ; ++k )
1799 { attr = subtree ( k , T ) ;
1800 switch ( nameofnode ( attr ) )
1801 { case P_ID :
1802 error
1803 { attr ,

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
shellswm/walk_4.c

DATE 5/23/89
TIME 4:42:27 pm

PAGE #
16/92

```

LINE # SOURCE TEXT
1801 /*MSG*/"illegal adapter pin attribute: ts"
1802 , syntostr ( textofnode ( attr ) )
1803 }
1804 break ;
1805 case N_ATTR :
1806 if
1807 ( !strcmp ( textofnode ( subtree ( 1 , attr ) ) ,
1808 C_trace_delay )
1809 )
1810 {
1811 set_text ( subtree ( 1 , attr ) , C_trace_delay ) ;
1812 if ( !td_seen )
1813 error
1814 ( subtree ( 1 , attr ) ,
1815 /*MSG*/"trace_delay adapter pin attribute respecified"
1816 ) ;
1817 else
1818 { delay = subtree ( 2 , attr ) ;
1819 if ( !nameofnode ( delay ) == P_ID_ )
1820 error
1821 ( delay ,
1822 /*MSG*/"trace_delay value is not numeric: ts"
1823 ) , syntostr ( textofnode ( delay ) )
1824 ) ;
1825 else
1826 { while
1827 ( !nameofnode ( delay ) == N_NEGATIVE &&
1828 !nameofnode ( subtree ( 1 , delay ) ) == N_NEGATIVE
1829 )
1830 delay = subtree ( 1 , subtree ( 1 , delay ) ) ;
1831 if ( !nameofnode ( delay ) == N_NEGATIVE )
1832 error
1833 ( delay ,
1834 /*MSG*/"negative trace_delay values illegal"
1835 ) ;
1836 else
1837 { val = check_sum ( delay ) ;
1838 if ( val > 65 )
1839 error
1840 ( delay ,
1841 /*MSG*/"trace_delay value too large: ts (must be no more than 65)"
1842 ) , syntostr ( textofnode ( delay ) )
1843 ) ;
1844 else
1845 { fkid = subtree ( 1 , T ) ;
1846 for
1847 ( td_seen = 1 ;
1848 td_seen <= kidcount ( fkid ) ;
1849 ++td_seen
1850 )
1851 { delay = subtree ( td_seen , fkid ) ;
1852 name = textofnode ( delay ) ;
1853 if
1854 ( ( name = spin_name_of ( name ) ) !=
1855 NULLSYM
1856 )
1857 if
1858 ( ( name = pin_number_of ( name ) ) !=
1859 NULLSYM
1860 )
1861 { num = pin_name_of ( name ) ;
1862 the -> pins [ num ] . trace_delay =
1863 val * 1000 + 0.5 ;
1864 }
1865 }
1866 td_seen = 1 ;
1867 }
1868 }
1869 }
1870 }
1871 }
1872 }
1873 }
1874 else
1875 error
1876 ( subtree ( 1 , attr ) ,
1877 /*MSG*/"illegal adapter pin attribute: ts (must be trace_delay)"
1878 ) , syntostr ( textofnode ( subtree ( 1 , attr ) ) )
1879 }
1880 break ;
1881 default :
1882 error
1883 ( attr ,
1884 /*MSG*/"internal error: bad adapter pin attribute"
1885 ) ;
1886 break ;
1887 }
1888 }
1889 tr_exit ( T ) ;
1890 } /* do_trace_delays */
1891
1892 /* end of trace_delay handling code */
1893
1894 /* start of pin name mapping handling code */
1895
1896 static void
1897 do_pin_mapping ( T , parent )
1898 tree T , parent ;
1899 {
1900 {
1901 done1 ,
1902 done2 ,
1903 kids1 ,
1904 kids2 ,
1905 k1 ,
1906 k2 ,
1907 sub1 ,
1908 sub2 ,
1909 val1 ,
1910 val2 ,
1911 max1 ,
1912 max2 ,
1913 n ,
1914 tree
1915 kids1 ,
1916 kids2 ,
1917 names1 ,
1918 names2 ,
1919 slook ,
1920 plook ,
1921 symbol

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswm/walk_4.c	DATE 5/23/89	PAGE # 17/93
LINE #	SOURCE TEXT			
1921	text1,			
1922	text2,			
1923	device_signal_name,			
1924	simulator_pin_name,			
1925	tr_entry (T) ,			
1926	namea1 = subtree (1 , T) ;			
1927	tr_entry (namea1) ,			
1928	kid1 = kidcount (namea1) ,			
1929	tr_exit (namea1) ;			
1930	namea2 = subtree (2 , T) ;			
1931	tr_entry (namea2) ,			
1932	kid2 = kidcount (namea2) ,			
1933	tr_exit (namea2) ;			
1934	done1 = done2 = 0 ;			
1935	k1 = k2 = 0 ;			
1936	sub1 = sub2 = 0 ;			
1937	while (! done1 && ! done2)			
1938	{ if (! sub1 val1 == max1)			
1939	{ if (k1 == kid1)			
1940	done1 = 1 ;			
1941	else			
1942	{ ++k1 ;			
1943	sub1 = 0 ;			
1944	}			
1945	else			
1946	val1 < max1 ? ++val1 : --val1 ;			
1947	if (! sub2 val2 == max2)			
1948	if (k2 == kid2)			
1949	done2 = 1 ;			
1950	else			
1951	{ ++k2 ;			
1952	sub2 = 0 ;			
1953	}			
1954	else			
1955	val2 < max2 ? ++val2 : --val2 ;			
1956	if (! done1 && ! done2)			
1957	{ if (sub1)			
1958	simulator_pin_name = gen_name (text1 , val1) ;			
1959	else			
1960	{ kid1 = subtree (k1 , namea1) ;			
1961	tr_entry (kid1) ;			
1962	switch (nameofnode (kid1))			
1963	{ case P_ID :			
1964	case P_NUM :			
1965	simulator_pin_name = textofnode (kid1) ;			
1966	break ;			
1967	case P_STR :			
1968	buf_string (syntostr (textofnode (kid1))) ;			
1969	simulator_pin_name = strtosys (buf) ;			
1970	break ;			
1971	case N_PIN_NAME :			
1972	sub1 = 1 ;			
1973	tr_entry (subtree (1 , kid1)) ;			
1974	text1 = textofnode (subtree (1 , kid1)) ;			
1975	tr_exit (subtree (1 , kid1)) ;			
1976	val1 = max1 = check_int (subtree (2 , kid1)) ;			
1977	if (kidcount (kid1) == 3)			
1978	max1 = check_int (subtree (3 , kid1)) ;			
1979	simulator_pin_name = gen_name (text1 , val1) ;			
1980	break ;			
1981	default :			
1982	error			
1983	(kid1 ,			
1984	/*LMS*/"internal error: bad simulator pin name"			
1985) ;			
1986	break ;			
1987	tr_exit (kid1) ;			
1988	}			
1989	if (sub2)			
1990	device_signal_name = gen_name (text2 , val2) ;			
1991	else			
1992	{ kid2 = subtree (k2 , namea2) ;			
1993	tr_entry (kid2) ;			
1994	switch (nameofnode (kid2))			
1995	{ case P_ID :			
1996	case P_NUM :			
1997	device_signal_name = textofnode (kid2) ;			
1998	break ;			
1999	case P_STR :			
2000	buf_string (syntostr (textofnode (kid2))) ;			
2001	device_signal_name = strtosys (buf) ;			
2002	break ;			
2003	case N_PIN_NAME :			
2004	sub2 = 1 ;			
2005	tr_entry (subtree (1 , kid2)) ;			
2006	text2 = textofnode (subtree (1 , kid2)) ;			
2007	tr_exit (subtree (1 , kid2)) ;			
2008	val2 = max2 = check_int (subtree (2 , kid2)) ;			
2009	if (kidcount (kid2) == 3)			
2010	max2 = check_int (subtree (3 , kid2)) ;			
2011	device_signal_name = gen_name (text2 , val2) ;			
2012	break ;			
2013	default :			
2014	error			
2015	(kid2 ,			
2016	/*LMS*/"internal error: bad device signal name"			
2017) ;			
2018	break ;			
2019	tr_exit (kid2) ;			
2020	}			
2021	slook = g_lookup (device_signal_name) ;			
2022	if			
2023	{ slook == NULLPTR			
2024	nameofnode (slook) != N_IN_PINS &&			
2025	nameofnode (slook) != N_IO_PINS &&			
2026	nameofnode (slook) != N_NC_PINS &&			
2027	nameofnode (slook) != N_OUT_PINS &&			
2028	nameofnode (slook) != N_POWER_PINS &&			
2029	nameofnode (slook) != N_GROUND_PINS			
2030	}			
2031	error			
2032	(kid2 ,			
2033	/*LMS*/"no such device signal name: %s"			
2034) , syntostr (device_signal_name)			
2035) ;			
2036	else			
2037	{ n = pin_name_of (device_signal_name) ;			
2038	slook = g_lookup (simulator_pin_name) ;			
2039	}			
2040				

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

F

DATE

5/23/89

PAGE #

TIME 4:42:27 pm

18/94

```

LINE # SOURCE TEXT
2041 if ( plook != NULLTREE && nameofnode ( plook ) == N_PIN_MAP )
2042 error
2043 ( kidl ,
2044 /*IMSC*/"simulator pin name is redeclared"
2045 , symtostr ( simulator_pin_name )
2046 )
2047 else if ( the -> pins [ n ] . alias != NULLSYM )
2048 error
2049 ( kidl ,
2050 /*IMSC*/"pin name mapping for device signal name is respecified: %s"
2051 , symtostr ( device_signal_name ) ,
2052 symtostr ( simulator_pin_name )
2053 )
2054 else
2055 { declare ( simulator_pin_name , parent ) ,
2056 the -> pins [ n ] . alias = simulator_pin_name ,
2057 }
2058 }
2059 }
2060 if ( ! done1 )
2061 error
2062 ( names2 ,
2063 /*IMSC*/"fewer device signal names than simulator pin names"
2064 )
2065 else if ( ! done2 )
2066 error
2067 ( names1 ,
2068 /*IMSC*/"fewer simulator pin names than device signal names"
2069 )
2070 }
2071 tr_exit ( T ) ,
2072 } /* do_pin_mapping */
2073
2074 /* end of pin name mapping handling code */
2075
2076 void
2077 walk_4 ( T )
2078 tree T ,
2079 /* the constraints: fourth and final walk */
2080 { ushort
2081 k ,
2082 kids ,
2083 len ,
2084 is ,
2085 pre_len ,
2086 post_len ,
2087 tr_entry ( T ) ,
2088 kids = kidcount ( T ) ,
2089 switch ( nameofnode ( T ) )
2090 { case N_INITIALIZE :
2091 k = 1
2092 if ( the -> pre_seq_length == 0 && the -> fb_seq_length == 0 )
2093 { do_reset_seq
2094 ( subtree ( 1 , T ) ,
2095 & the -> rise_fb ,
2096 & the -> pre_seq_length ,
2097 & the -> fb_seq_length ,
2098 & the -> post_seq_length
2099 )
2100 if ( the -> fb_seq_length && ! the -> has_feedback_pins )
2101 error
2102 ( subtree ( 1 , subtree ( 1 , T ) ) ,
2103 /*IMSC*/"must have a feedback signal to use feedback sequences"
2104 )
2105 if ( ! the -> fb_seq_length && the -> has_feedback_pins )
2106 error
2107 ( subtree ( 1 , subtree ( 1 , T ) ) ,
2108 /*IMSC*/"must use a feedback sequence to have a feedback signal"
2109 )
2110 ++k
2111 }
2112 for ( , k ( = kids , ++k )
2113 { do_reset_seq
2114 ( subtree ( k , T ) ,
2115 & is ,
2116 & pre_len ,
2117 & len ,
2118 & post_len
2119 )
2120 if
2121 { /*pre_len &&
2122 the -> pre_seq_length &&*/
2123 pre_len != the -> pre_seq_length
2124 }
2125 { buf_instead ( pre_len , the -> pre_seq_length ) ,
2126 error
2127 ( subtree ( 1 , subtree ( k , T ) ) ,
2128 /*IMSC*/"pre-feedback initialization sequence length mismatch: %s"
2129 , buf
2130 )
2131 }
2132 if
2133 { /*len && the -> fb_seq_length &&*/
2134 len != the -> fb_seq_length
2135 }
2136 { buf_instead ( len , the -> fb_seq_length ) ,
2137 error
2138 ( subtree ( 1 , subtree ( k , T ) ) ,
2139 /*IMSC*/"feedback sequence length mismatch: %s"
2140 , buf
2141 )
2142 }
2143 if ( len && the -> fb_seq_length && is != the -> rise_fb )
2144 error
2145 ( subtree ( 1 , subtree ( k , T ) ) ,
2146 /*IMSC*/"feedback condition mismatch: %s (all feedback conditions must be the same)"
2147 , symtostr ( ( symbol ) ( is ? C_rise : C_fall ) )
2148 )
2149 }
2150 if
2151 { /*post_len && the -> post_seq_length &&*/
2152 post_len != the -> post_seq_length
2153 }
2154 { buf_instead ( post_len , the -> post_seq_length ) ,
2155 error
2156 ( subtree ( 1 , subtree ( k , T ) ) ,
2157 /*IMSC*/"post-feedback initialization sequence length mismatch: %s"
2158 , buf
2159 )
2160 }
2161 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

shellswm/walk_4.c

DATE 5/23/89

PAGE #

TIME 4:42:27 pm

19/95

```
LINE # SOURCE TEXT
2161 break ;
2162 case M_DELAY :
2163     the -> delays_coun = 1 ;
2164     for ( k = 1 ; k <= kids ; ++k )
2165         do_from_delay ( subtree ( k , T ) ) ;
2166     break ;
2167 case M_ADAPTER :
2168     for ( k = 1 ; k <= kids ; ++k )
2169         do_trace_delays ( subtree ( k , T ) ) ;
2170     break ;
2171 case M_PIN_MAP :
2172     for ( k = 1 ; k <= kids ; ++k )
2173         do_ps_mapping ( subtree ( k , T ) , T ) ;
2174     break ;
2175     .default : break ;
2176 }
2177 tr_exit ( T ) ;
2178 /* walk_4 */
2179
2180 /* end of Constrainer (walk_4) for EDML Processor */
2181
```

849

5,353,243

850

Copyright 1989

Modeling Systems

SOURCE PROGRAM
shellswc/Pr_comm.c

DATE 5/23/89

PAGE #

TIME 1:21:06 pm

1/1

SOURCE TEXT

```

1 // SCSS ID: Pr_comm.c rev 3.2, 5/9/89 at 15:45:04
2
3 /*
4  * Processor for EMU - COMMON PORTION
5  */
6
7 #include <stdio.h>
8 #include "common.h"
9 #include "EMU.h"
10 #include "strng.h"
11 #include "Sparse.h"
12 #include "Tree.h"
13 #include "Xconst.h"
14 #include "device.h"
15
16 #define ERROR_MSG 1
17 #define WARNING_MSG 2
18
19 extern char *lexical_pass ( ) ;
20 extern void free_input ( ) ;
21 extern struct device *device_pass ( ) ;
22
23 static char
24     *program , /* pointer to name of program */
25     *input , /* input file name */
26 static ushort
27     lineno , /* source line number */
28     errs , /* fatal error counter */
29     c_flag , /* comp power/ground pins flag */
30     d_flag , /* device print flag */
31     g_flag , /* device statistics flag */
32     l_flag , /* list grammar flag */
33     m_flag , /* print mapping flag */
34     o_flag , /* no write flag */
35     p_flag , /* print tree flag */
36     s_flag , /* string statistics flag */
37     t_flag , /* trace constraints flag */
38     w_flag , /* write device flag */
39
40 /*VARARGS2*/
41 void
42 lm_Error ( err , format , arg , arg2 )
43     ushort err ;
44     char
45         *format ,
46         *arg ,
47         *arg2 ;
48 /* reports error indicated in format and arg to stderr */
49 /* if global line number is nonzero, also prints file name, line number */
50 /* increments global error count by err */
51 { char arbuf [ 256 ] ;
52   ( void ) sprintf ( arbuf , "%s\n", program ) ;
53   *arbuf = '\0' ;
54   if ( lineno )
55   { char num [ 12 ] ;
56     ( void ) sprintf ( arbuf + strlen ( arbuf ) , "%s\n", input ) ;
57     ( void ) sprintf ( num , "%d", lineno ) ;
58     ( void ) sprintf ( arbuf + strlen ( arbuf ) , "line %s: ", num ) ;
59   }
60   ( void ) sprintf ( arbuf + strlen ( arbuf ) , format , arg , arg2 ) ;
61   lm_queue_message ( ERROR_MSG , "%s", arbuf ) ;
62   errs += err ;
63   /* lm_Error */
64 }
65
66 /*VARARGS2*/
67 void
68 lm_Warning ( format , arg , arg2 )
69     char
70         *format ,
71         *arg ,
72         *arg2 ;
73 /* reports error indicated in format and arg to stderr */
74 /* if global line number is nonzero, also prints file name, line number */
75 /* increments global error count by err */
76 { char arbuf [ 256 ] ;
77   ( void ) sprintf ( arbuf , "%s\n", program ) ;
78   *arbuf = '\0' ;
79   if ( lineno )
80   { char num [ 12 ] ;
81     ( void ) sprintf ( arbuf + strlen ( arbuf ) , "%s\n", input ) ;
82     ( void ) sprintf ( num , "%d", lineno ) ;
83     ( void ) sprintf ( arbuf + strlen ( arbuf ) , "line %s: ", num ) ;
84     ( void ) sprintf ( arbuf + strlen ( arbuf ) , "warning: " ) ;
85   }
86   ( void ) sprintf ( arbuf + strlen ( arbuf ) , format , arg , arg2 ) ;
87   lm_queue_message ( WARNING_MSG , "%s", arbuf ) ;
88   /* lm_Warning */
89 }
90
91 ushort
92 lm_Errors ( )
93 /* returns global error count */
94 { return errs ;
95   /* lm_Errors */
96 }
97
98 static void
99 init_flags ( )
100 { c_flag =
101   d_flag =
102   g_flag =
103   l_flag =
104   m_flag =
105   o_flag =
106   p_flag =
107   s_flag =
108   t_flag =
109   w_flag =
110   0 ;
111   /* init_flags */
112 }
113
114 void
115 lm_Init_vars ( name )
116     char *name ;
117 { program = name ;
118   input = "" ;
119   lineno = 0 ;
120   errs = 0 ;
121   init_flags ( ) ;
122   /* lm_Init_vars */
123 }
124
125 void

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswc/Pr_comm.c	DATE 5/23/89	PAGE # 2/2
LINE #	SOURCE TEXT			
121	lm_hset_c_flag ()			
122	/* sets c_flag */			
123	{ c_flag = 1 ;			
124	/* lm_hset_c_flag */			
125	}			
126	void			
127	lm_hset_d_flag ()			
128	/* sets d_flag */			
129	{ d_flag = 1 ;			
130	/* lm_hset_d_flag */			
131	}			
132	void			
133	lm_hset_g_flag ()			
134	/* sets g_flag */			
135	{ g_flag = 1 ;			
136	/* lm_hset_g_flag */			
137	}			
138	void			
139	lm_hset_l_flag ()			
140	/* sets l_flag */			
141	{ l_flag = 1 ;			
142	/* lm_hset_l_flag */			
143	}			
144	void			
145	lm_hset_m_flag ()			
146	/* sets m_flag */			
147	{ m_flag = 1 ;			
148	/* lm_hset_m_flag */			
149	}			
150	void			
151	lm_hset_n_flag ()			
152	/* sets n_flag */			
153	{ n_flag = 1 ;			
154	/* lm_hset_n_flag */			
155	}			
156	void			
157	lm_hset_p_flag ()			
158	/* sets p_flag */			
159	{ p_flag = 1 ;			
160	/* lm_hset_p_flag */			
161	}			
162	void			
163	lm_hset_s_flag ()			
164	/* sets s_flag */			
165	{ s_flag = 1 ;			
166	/* lm_hset_s_flag */			
167	}			
168	void			
169	lm_hset_t_flag ()			
170	/* sets t_flag */			
171	{ t_flag = 1 ;			
172	/* lm_hset_t_flag */			
173	}			
174	void			
175	lm_hset_v_flag ()			
176	/* sets v_flag */			
177	{ v_flag = 1 ;			
178	/* lm_hset_v_flag */			
179	}			
180	ushort			
181	lm_hc_flag_on ()			
182	/* Returns state of c_flag */			
183	{ return c_flag ;			
184	/* lm_hc_flag_on */			
185	}			
186	ushort			
187	lm_hd_flag_on ()			
188	/* Returns state of d_flag */			
189	{ return d_flag ;			
190	/* lm_hd_flag_on */			
191	}			
192	ushort			
193	lm_hg_flag_on ()			
194	/* Returns state of g_flag */			
195	{ return g_flag ;			
196	/* lm_hg_flag_on */			
197	}			
198	ushort			
199	lm_hl_flag_on ()			
200	/* Returns state of l_flag */			
201	{ return l_flag ;			
202	/* lm_hl_flag_on */			
203	}			
204	ushort			
205	lm_hm_flag_on ()			
206	/* Returns state of m_flag */			
207	{ return m_flag ;			
208	/* lm_hm_flag_on */			
209	}			
210	ushort			
211	lm_hn_flag_on ()			
212	/* Returns state of n_flag */			
213	{ return n_flag ;			
214	/* lm_hn_flag_on */			
215	}			
216	ushort			
217	lm_hp_flag_on ()			
218	/* Returns state of p_flag */			
219	{ return p_flag ;			
220	/* lm_hp_flag_on */			
221	}			
222	ushort			
223	lm_hs_flag_on ()			
224	/* Returns state of s_flag */			
225	{ return s_flag ;			
226	/* lm_hs_flag_on */			
227	}			
228	ushort			
229	lm_hs_flag_on ()			
230	/* Returns state of t_flag */			
231	{ return t_flag ;			
232	/* lm_hs_flag_on */			
233	}			
234	ushort			
235	lm_hv_flag_on ()			
236	/* Returns state of v_flag */			
237	{ return v_flag ;			
238	/* lm_hv_flag_on */			
239	}			
240	void			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM shellswc/Pr_comm.c	DATE 5/23/89	PAGE # 3/3
LINE #	SOURCE TEXT			
241	/* in_Get_input (str)			
242	/* sets input file name to the passed string */			
243	/* you must not change the string's contents */			
244	input = str ;			
245	/* in_Get_input */			
246	{			
247	char *			
248	in_Get_input ()			
249	/* returns input file name */			
250	{ return input ;			
251	/* in_Get_input */			
252	{			
253	void			
254	in_Get_lineno (num)			
255	/* sets global line number to passed number */			
256	{ lineno = num ;			
257	/* in_Get_lineno */			
258	{			
259	/* returns global line number */			
260	{ return lineno ;			
261	/* in_Get_lineno */			
262	{			
263	/* end of Processor for INEL CORON PORTION */			
264	{			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/cpu.c

DATE 5/23/89
TIME 4:41:13 pm

PAGE #
1/1

```

1  /* SCCS ID: cpu.c rev 3.1, 4/24/89 at 07:48:20 */
2  /*.....*/
3  /* CPU routines */
4  /*.....*/
5
6  #include <math.h>
7  #include "common.h"
8  #include "lm_diags.h"
9  #include "vr_tx.h"
10 #include "cpu.h"
11 #include "modeler_exta.h"
12
13 int hardware_reset(), flash();
14
15 cpu_diag_disp(parent_menu)
16 LM_DIAG_MENU *parent_menu;
17 {
18     static LM_DIAG_MENU_ITEM menu_list[] =
19     {
20         {
21             "i",
22             "Flash leds",
23             flash,
24             LM_DIAG_utility,
25             LM_DIAG_sull,
26             0
27         },
28         {
29             "r",
30             "HARDWARE RESET",
31             hardware_reset,
32             LM_DIAG_utility,
33             LM_DIAG_sull,
34             0
35         },
36     },
37     static LM_DIAG_MENU menu =
38     {
39         "CPU UTILITIES", /* not used */
40         sizeof(menu_list) / sizeof(LM_DIAG_MENU_ITEM),
41         0,
42         menu_list
43     },
44     menu.title = parent_menu->
45     menu_items[parent_menu->current_selection].menu_text;
46     return lm_display_menu(&menu);
47 }
48
49 #define LEDS 0x00
50 #define REC_LEDS 0x00
51
52 flash()
53 {
54     register char *cr = (char *)CPU_MISC_CONTROL, startstate, nextstate;
55     register long flag;
56     register int delay;
57
58     startstate = *cr & ~LEDS;
59     for (delay = 500; delay > 0; delay = (2 * delay)/3) {
60         for (flag = 0; flag <= 8; ++flag) {
61             nextstate = startstate | (((flag << 5) & LEDS) ^ REC_LEDS);
62             *cr = nextstate;
63             lm_delay(delay);
64         }
65     }
66     return SUCCESS;
67 }
68
69 hardware_reset()
70 {
71     lm_message("HARDWARE RESET...\n");
72     lm_delay(100);
73     ((cpu_control_reg_struct *)CPU_CONTROL_REG)->suicide = 1;
74     /* See you later ..... */
75     while(1)
76     {
77     }
78 }
79
80 set_fault_led()
81 {
82     ((cpu_control_reg_struct *)CPU_CONTROL_REG)->fault_led = LED_ON;
83 }
84
85 clear_fault_led()
86 {
87     ((cpu_control_reg_struct *)CPU_CONTROL_REG)->fault_led = LED_OFF;
88 }

```

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

diags/debugger.c

DATE 5/23/89

PAGE #

TIME 4:41:13 pm

1/2

```

1  /* SCCS_ID: debugger.c rev 3.1, 4/24/89 at 07:48:22 */
2
3  #include "common.h"
4
5  #define is_hex_sum(c) (((c) >= '0') && ((c) <= '9')) \
6                      || ((c) >= 'a') && ((c) <= 'f')) \
7                      || ((c) >= 'A') && ((c) <= 'F'))
8
9  #define ascii_to_hex_byte(c) (((c) >= '0') && ((c) <= '9'))? \
10 (char)((c) - '0') : (((c) >= 'a') && ((c) <= 'f'))? \
11 (char)((c) - 10 - 'a') : (char)((c) + 10 - 'A'))
12
13 #define ctrl(c) ('c' & 0x1f)
14
15 debugger()
16 {
17     char c;
18     int (*start)();
19     u_long accumulator = 0;
20     u_long address = 0;
21     int building_accum = 0;
22     int line, l, lcount;
23     char buffer[128];
24
25     lm_banner(" ");
26     while (1) {
27         c = lm_get_key();
28         if (is_hex_sum(c)) {
29             lm_banner("c", c);
30             if (building_accum == 0) {
31                 lcount = 0;
32                 accumulator = 0;
33             }
34             lcount++;
35             accumulator = (accumulator << 4) + ascii_to_hex_byte(c);
36             if (building_accum > 8) {
37                 building_accum = 0;
38                 sprintf(buffer, "%08x", accumulator);
39                 lm_banner(buffer);
40             }
41         } else if (c == '\b') {
42             if (lcount) {
43                 lm_banner("\b\b");
44                 --lcount;
45             }
46             if (building_accum) {
47                 accumulator >>= 4;
48                 --building_accum;
49             }
50         } else {
51             if (c == ' ') {
52                 lm_banner("tc", c);
53             }
54             switch (c) {
55                 case '0':
56                     lm_banner("uit\n");
57                     return;
58                 case '\n':
59                     lm_banner("\n");
60                     break;
61                 case '\r':
62                     lm_banner("\n");
63                     break;
64                 case 'l':
65                     lm_banner("\n");
66                     if (building_accum)
67                         address = accumulator;
68                     break;
69                 case 'm':
70                     lm_banner("\n");
71                     if (building_accum)
72                         address = accumulator;
73                     for (line = 0; line < 20; ++line) {
74                         sprintf(buffer, "%08x:", address);
75                         lm_banner(buffer);
76                         for (l = 0; l < 8; ++l) {
77                             sprintf(buffer, " %04x",
78                                     "(u_short *)
79                                     address + 0x1fff);
80                             lm_banner(buffer);
81                             address += 2;
82                         }
83                         lm_banner("\n");
84                     }
85                     break;
86                 case 'r':
87                     lm_banner("\n");
88                     if (building_accum)
89                         address = accumulator;
90                     sprintf(buffer, "%08x: %04x ",
91                             address,
92                             "(u_short *)
93                             address + 0xffff);
94                     lm_banner(buffer);
95                     break;
96                 case 'w':
97                     "(u_short *)address = accumulator;
98                     building_accum = 0;
99                     /* fall through */
100                 case '-':
101                     lm_banner("\n");
102                     if (building_accum)
103                         address = accumulator;
104                     else
105                         address += 2;
106                     sprintf(buffer, "%08x: %04x ",
107                             address,
108                             "(u_short *)
109                             address + 0xffff);
110                     lm_banner(buffer);
111                     break;
112                 case '.':
113                     lm_banner("\n");
114                     if (building_accum)
115                         address = accumulator;
116                     else
117                         address += 2;
118                     sprintf(buffer, "%08x: %04x ",
119                             address,
120                             "(u_short *)

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/debugger.c	DATE 5/23/89 TIME 4:41:13 pm	PAGE # 2/3
LINE #	SOURCE TEXT			
121	address & 0xffff);			
122	lm_banner(buffer);			
123	break;			
124	case 'h':			
125	lm_banner("\n");			
126	print_help();			
127	break;			
128	default:			
129	lm_banner("\n");			
130	break;			
131	}			
132	building_accum = 0;			
133	lm_banner("\n");			
134	}			
135	}			
136	}			
137	}			
138	static char *help_list[] = {			
139	"LM DIAGNOSTICS DEBUGGER COMMAND SUMMARY",			
140	"			
141	"			
142	" (a)l set current address to (a)",			
143	" (a)m display 320 bytes of memory from (a) or current address",			
144	" (a)r display memory word at (a) or current address",			
145	" (a)w write (a) to current address and increment",			
146	" (a)+ display memory word at (a) or next address",			
147	" (a)- display memory word at (a) or previous address",			
148	" q quit debugger",			
149	" ? or h display help summary",			
150	0			
151	};			
152	static			
153	print_help()			
154	{			
155	char *c = help_list;			
156	char buffer[128];			
157	while (*c) {			
158	sprintf(buffer, "%s\n", *c++);			
159	lm_banner(buffer);			
160	}			
161	}			
162	}			
163	}			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/diag_bus.c

DATE 5/23/89
TIME 4:41:14 pm

PAGE #
1/4

```

1  /* SCCS ID: diag_bus.c rev 3.1, 4/24/89 at 07:48:25 */
2
3  /*
4  ** bus.c
5  ** bus error handler
6  ** It is more complicated than it needs to be,
7  ** we need to use this to determine if a PAC
8  ** is present...
9  */
10 #include "common.h"
11 #include "cpu.h"
12 #include "cpu_vec.h"
13 #include "mod_err.h"
14 #include "svaran.h"
15 #include "lm_rd_wr.h"
16 #include "bus.h"
17 static char buff( 200 );
18 extern bus_iar();
19 u_long setup_ivt_bus();
20 extern u_short lm_svaran_access();
21 extern void reset_cpu();
22 void bus_error_txz();
23 void output_routine();
24 u_short address_is_map();
25 static u_char ignore_low_address;
26 static u_char ignore_hi_address;
27 u_char ignore_ssw_error;
28 static u_char ssw_error;
29 static char debug_bus = 1;
30 /*
31 ** Bus error handler
32 */
33 diag_bus_error()
34 {
35     extern STACK_FRAME *bus_error_address;
36     STACK_FRAME *stack_frame = bus_error_address;
37     u_long err;
38     int i;
39     extern u_long Diag_bus_err_addr;
40     char buffer[80];
41
42     if( address_is_map( stack_frame->address ) == SUCCESS)
43     {
44         if( ignore_ssw_error == TRUE )
45             if( ignore_low_address >= stack_frame->address &&
46                 ignore_hi_address <= stack_frame->address)
47             {
48                 if( stack_frame->spec_status_word & 0x0100)
49                 {
50                     stack_frame->spec_status_word ^= 0x0100;
51                     ssw_error = TRUE;
52                     return;
53                 }
54             }
55
56             if( lm_svaran_access( (char *) &stack_frame->address,
57                                 BUS_ADDR, (u_long)sizeof BUS_ADDR, MEMORY_WRITE, &err) == FAILURE)
58             {
59                 reset_cpu(SUICIDE);
60             }
61             if( lm_svaran_access( (char *) &stack_frame->spec_status_word,
62                                 SSW_REG, (u_long)sizeof SSW_REG, MEMORY_WRITE, &err) == FAILURE)
63             {
64                 reset_cpu(SUICIDE);
65             }
66     }
67     /* we crashed
68     */
69     Diag_bus_err_addr = (int)(stack_frame->address);
70     stack_frame->pc = (u_long)intr;
71     stack_frame->vector = 0x0fff; /*pretend we got a 4 word stack frame */
72 }
73
74 #define BUS_ERR_VECT *((int *)0)
75
76 install_diag_bus_error()
77 {
78     int diag_bus_iar();
79
80     BUS_ERR_VECT = (int)diag_bus_iar;
81 }

```

Copyright 1989 Logic Modeling Systems		FILE diags/diag_setjmp.s	DATE 5/23/89	PAGE # 1/5
LINE #		TEXT	TIME 4:41:14 pm	
1		SOCS ID: diag_setjmp.s rev 3.1. 4/24/89 at 07:48:27		
2		.text		
3		.globl _diag_setjmp		
4		_diag_setjmp:		
5		movl sp(4),a0		
6		movl sp(0),a00		
7		movl _la_current_depth,a0(4) Vital menu state		
8		clrl a0(8)		
9		moveml d2/d3/d4/d5/d6/d7/a2/a3/a4/a5/a6/a7,a0(0xc)		
10		clrl d0		
11		rtl		
12		.globl _diag_longjmp		
13		_diag_longjmp:		
14		movl sp(4),a0		
15		movl a0(4),_la_current_depth		
16		movl sp(8),d0		
17		bne NOTZERO		
18		moveq \$1,d0		
19		NOTZERO:		
20		movl a00,a1		
21		moveml a0(0xc),d2/d3/d4/d5/d6/d7/a2/a3/a4/a5/a6/a7		
22		addq1 \$4,sp		
23		jmp a1		
24				

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/fit.c

DATE 5/23/89
TIME 4:41:14 pm

PAGE #
1/6

```

LINE # SOURCE TEXT
1  /* SCCL_ID: fit.c rev 3.1, 4/24/89 at 07:48:29 */
2  /*-----*/
3  1/23/89
4  fit.c
5  Routine from Numerical Recipes in C
6  Calculates best fit line to data.
7
8  Given a set of points x[1..ndata], y[1..ndata] with standard
9  deviations sigy[1..ndata], fit them to a straight line
10 y = a + b*x by minimizing chi**2. Returned are a, b, and their
11 respective probable uncertainties siga and sigb, the
12 chi-square chi2, and the goodness-of-fit probability q (that
13 the fit would have chi**2 this large or larger). If mwt = 0
14 as input, then the standard deviations are assumed to be
15 unavailable: q is returned as 1.0 and the normalization of
16 chi2 is to unit standard deviation on all points.
17 Note: y is the independent variable with error variance sigy
18 and x is the dependent variable (known exactly).
19
20 Inputs:
21 x[1..ndata]: measured time delay
22 y[1..ndata]: threshold value
23 ndata: number of points
24 sigy[1..ndata]: array of standard deviations
25
26 Returns:
27 *a, *b: coeff of line y = a + b*x
28 *sig_a, *sig_b: probable uncertainties of a, b
29 *chi2: chi square
30 *q: goodness of fit
31
32 /*-----*/
33 #include <stdio.h>
34 #include <math.h>
35
36 static float sqarray;
37 #define SQR(a) (sqarray*(a)*sqarray*sqarray)
38
39 /*UNUSED*/
40 void fit(x,y,sigy,mwt,a,b,sig_a,sig_b,chi2,q)
41 float x[],y[],sigy[],*a,*b,*sig_a,*sig_b,*chi2,*q;
42 unsigned long mwt;
43 unsigned long ndata;
44 {
45     int i;
46     float t,sxosa,sx=0.0,sy=0.0,at1=0.0,ss,sigdat;
47 #ifdef LOG_DEFINED
48     float wt;
49     float gammq();
50 #endif LOG_DEFINED
51
52     *b = 0.0;
53 #ifdef LOG_DEFINED
54     if(mwt) /* accumulate sums */
55     {
56         ss = 0.0;
57         for(i = 1; i <= ndata; i++) /* ...with weights */
58         {
59             wt = 1.0 / SQR(sigy[i]);
60             ss += wt;
61             sx += x[i] * wt;
62             sy += y[i] * wt;
63         }
64     }
65     else
66 #endif LOG_DEFINED
67     {
68         for(i = 1; i <= ndata; i++) /* ...without weights */
69         {
70             sx += x[i];
71             sy += y[i];
72         }
73         ss = ndata;
74     }
75     sxosa = sx / ss;
76 #ifdef LOG_DEFINED
77     if(mwt)
78     {
79         for(i = 1; i <= ndata; i++)
80         {
81             t = (x[i] - sxosa) / sigy[i];
82             st2 += t * t;
83             *b += t * y[i] / sigy[i];
84         }
85     }
86     else
87 #endif LOG_DEFINED
88     {
89         for(i = 1; i <= ndata; i++)
90         {
91             t = x[i] - sxosa;
92             st2 += t * t;
93             *b += t * y[i];
94         }
95     }
96     *b /= st2; /* Solve for a, b, sig_a, sig_b */
97     *a = (sy - sx * (*b)) / ss;
98     *sig_a = sqrt(1.0 + sx * sx / (ss * st2)) / ss;
99     *sig_b = sqrt(1.0 / st2);
100     *chi2 = 0.0; /* calculate chi squared */
101 #ifdef LOG_DEFINED
102     if(mwt)
103     {
104         for(i = 1; i <= ndata; i++)
105             *chi2 += SQR((y[i] - (*a) - (*b) * x[i]) / sigy[i]);
106         *q = gammq(0.5 * (ndata - 2), 0.5 * (*chi2));
107     }
108     else
109 #endif LOG_DEFINED
110     {
111         for(i = 1; i <= ndata; i++)
112             *chi2 += SQR(y[i] - (*a) - (*b) * x[i]);
113         *q = 1.0;
114         sigdat = sqrt((*chi2) / (ndata - 2));
115         *sig_a = sigdat;
116         *sig_b = sigdat;
117     }
118 }
119
120 #ifdef LOG_DEFINED

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/fit.c	DATE 5/23/89 TIME 4:41:14 pm	PAGE # 2/7
LINE #	SOURCE TEXT			
121	float gammc(a, x) /* returns incomplete gamma function */			
122	float a, x; /* Q(a, x) = 1 - P(a, x) */			
123	{			
124	float gmmear, gmmcf, gla;			
125	void gcf(), gear(), arerror();			
126	{			
127	if(x < 0.0 a <= 0.0)			
128	arerror("Invalid arguments in routine GAMMC");			
129	if(x < (a + 1.0)) /* use the series representation */			
130	{			
131	gear(&gmmear, a, x, &gla);			
132	return(1.0 - gmmear); /* and take its complement */			
133	}			
134	else /* Use the continued fraction rep. */			
135	{			
136	gcf(&gmmcf, a, x, &gla);			
137	return(gmmcf);			
138	}			
139	}			
140	#define ITMAX 100			
141	#define EPS 1.0e-7			
142	void gear(gmmear, a, x, gla)			
143	float a, x, *gmmear, *gla;			
144	{			
145	int n;			
146	float sum, del, ap;			
147	float gammc();			
148	void arerror();			
149	{			
150	printf("gear\n");			
151	printf("a = %f, x = %f\n", a, x);			
152	{			
153	gla = gammc(a);			
154	if(x <= 0.0)			
155	{			
156	if(x < 0.0)			
157	arerror("x less than 0 in routine GEAR");			
158	gmmear = 0.0;			
159	return;			
160	}			
161	else			
162	{			
163	ap = a;			
164	del = sum = 1.0 / a;			
165	for(n = 1; n <= ITMAX; n++)			
166	{			
167	ap += 1.0;			
168	del += x / ap;			
169	sum += del;			
170	if(fabs(del) < fabs(sum) * EPS)			
171	{			
172	gmmear = sum * exp(-x + a * log(x) - (*gla));			
173	return;			
174	}			
175	}			
176	arerror("a too large, ITMAX too small in routine GEAR");			
177	return;			
178	}			
179	void gcf(gmmcf, a, x, gla)			
180	float a, x, *gmmcf, *gla;			
181	{			
182	int n;			
183	float gold=0.0, g, fac=1.0, bl=1.0;			
184	float b0=0.0, anf, ans, an, al, a0=1.0;			
185	float gammc();			
186	void arerror();			
187	{			
188	printf("gcf\n");			
189	printf("a = %f, x = %f\n", a, x);			
190	{			
191	gla = gammc(a);			
192	if(x <= 0.0)			
193	{			
194	if(x < 0.0)			
195	arerror("x less than 0 in routine GCF");			
196	gmmcf = 0.0;			
197	return;			
198	}			
199	else			
200	{			
201	an = (float) a;			
202	ans = an - a;			
203	a0 = (al + a0 + ans) * fac;			
204	b0 = (bl + b0 + ans) * fac;			
205	anf = an * fac;			
206	al = x * a0 + anf * al;			
207	bl = x * b0 + anf * bl;			
208	if(al)			
209	{			
210	fac = 1.0 / al;			
211	g = bl * fac;			
212	if(fabs((g - gold) / g) < EPS)			
213	{			
214	gmmcf = exp(-x + a * log(x) - (*gla)) * g;			
215	return;			
216	}			
217	gold = g;			
218	}			
219	}			
220	arerror("a too large, ITMAX too small in routine GCF");			
221	return;			
222	}			
223	float gammc(xx)			
224	float xx;			
225	{			
226	double x, tmp, ser;			
227	static double cof[6] = {76.18009173, -86.50532033, 24.01409822,			
228	-1.231739516, 0.120858003e-2, -0.536182e-5};			
229	{			
230	int j;			
231	x = xx - 1.0;			
232	tmp = x + 5.5;			
233	tmp = (x + 0.5) * log(tmp);			
234	ser = 1.0;			
235	for(j = 0; j <= 5; j++)			
236	{			
237	x += 1.0;			
238	ser += cof[j] / x;			
239	}			
240	printf("log(gf)\n", 2.50662827465 * ser);			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/fit.c

DATE 5/23/89
TIME 4:41:14 pm

PAGE #
3/8

LINE # SOURCE TEXT

```
241 return(-tmp + log(2.50662827465 * ser));  
242 }  
243  
244 void arerror(error_text)  
245 char error_text[];  
246 {  
247     void exit();  
248     fprintf(stderr,"Numerical Recipes run-time error...\n");  
249     fprintf(stderr,"%s\n",error_text);  
250     fprintf(stderr,"...now exiting to system...\n");  
251     exit(1);  
252 }  
253 #endif LOG_DEFINED
```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/hex.c	DATE 5/23/89	PAGE # 1/9
LINE #	SOURCE TEXT			
1	/* SCDS_ID: hex.c rev 3.1, 4/24/89 at 07:48:32 */			
2	/*			
3	** MAIN.C			
4	** This file contains the startup code for the tasks.			
5	** Tasks and resources begin here			
6	*/			
7	#include <stdio.h>			
8	#include "common.h"			
9	#include "cpu.h"			
10	#include "task.h"			
11	#include "lance.h"			
12	#include "and_err.h"			
13	#include "nvram.h"			
14	#include "ls_rd_wr.h"			
15	/*			
16	** The globals			
17	*/			
18	extern unsigned long timer_semaphore, rcv_lance_pkt_semaphore,			
19	extern int key_hit,			
20	int play_semaphore, malloc_semaphore,			
21	/*			
22	** The BSP creates the main task, and looks for the entry point			
23	** named main			
24	*/			
25	u_long auto_start = 0;			
26	BOOT_STRUCT boot;			
27	main()			
28	{			
29	void serial_task();			
30	void output_routine();			
31	void transmit_task();			
32	void housekeeping_task();			
33	int modem_task();			
34	void diags();			
35	extern void set_boot();			
36	extern u_short ls_nvram_access();			
37	void lamp_receive_task();			
38	u_short write_lance_car(), read_lance_car();			
39	int err;			
40	extern u_long modem_inet,			
41	extern end,			
42	unsigned long error;			
43	/*			
44	** This routine creates the partition used in the system.			
45	** The only usage is by malloc and friends.			
46	*/			
47	create_malloc_partition(&end, (u_long)CPU_RAM_SIZE - (u_long)&end);			
48	printf("\n1900 Diagnostics loaded.\n");			
49	if(ls_nvram_access((char *) &boot, BOOT, sizeof BOOT, MEMORY_READ,			
50	&error) == FAILURE)			
51	{			
52	output_routine("\nFailed to read NVRam");			
53	}			
54	modem_inet = boot.modem_internet_address;			
55	modem_state = RUNNING_DIAG;			
56	(void)ls_nvram_access(&modem_state, MODELER_STATE,			
57	sizeof MODELER_STATE, MEMORY_WRITE, (u_long *) &error);			
58	/*			
59	** initialize the files			
60	*/			
61	fifo_initialize();			
62	modem_init(); /* init modem queues */			
63	/*			
64	key_hit = ac_create(0, 1, &err);			
65	if(&err) sys_out("\nError creating event flag for checking.");			
66	/*			
67	** The malloc semaphore controls access to the malloc and free			
68	** functions. This avoids re-entrancy problems associated with			
69	** a multi-tasking environment.			
70	*/			
71	malloc_semaphore = ac_create(1, 0, &err);			
72	if (&err) output_routine("\nError creating malloc semaphore");			
73	/*			
74	** end of play interrupt semaphore			
75	*/			
76	play_semaphore = ac_create(0, 1, &err);			
77	if(&err) sys_out("\nError creating event flag for EOP interrupt.");			
78	/*			
79	** create semaphore for lance rev 1.2			
80	** 0 = init value			
81	** 0 = priority order			
82	*/			
83	rcv_lance_pkt_semaphore = ac_create(0, 0, &err);			
84	if(&err) output_routine("\nError creating rcv lance pkt semaphore");			
85	/*			
86	if (get_lance_ready_to_go() != SUCCESS) {			
87	output_routine("\nError getting the LANCE ready to go.");			
88	ac_timespend(0, 0, &err);			
89	}			
90	/*			
91	if (start_lance() != SUCCESS) {			
92	output_routine("\nError starting the LANCE.");			
93	ac_timespend(0, 0, &err);			
94	}			
95	/*			
96	** spin of the tasks			
97	*/			
98	ac_tcreate(housekeeping_task, HOUSEKEEPING_TASK_ID,			
99	DIAG_HOUSEKEEPING_TASK_PRI, &err);			
100	if(&err) output_routine("\nError creating housekeeping task.");			
101	ac_tcreate(modem_task,			
102	KEYBOARD_TASK_ID, DIAG_KEYBOARD_TASK_PRI, &err);			
103	if(&err) sys_out("\nError creating modem task.");			
104	/*			
105	ac_tcreate(transmit_task,			
106	TRANSMIT_TASK_ID, DIAG_TRANSMIT_TASK_PRI, &err);			
107	if(&err) output_routine("\nError creating transmit task.");			
108	/*			
109	ac_tcreate(serial_task,			
110	SERIAL_TASK_ID, DIAG_SERIAL_TASK_PRI, &err);			
111	if(&err) output_routine("\nError creating serial task.");			
112	/*			
113	** spin of the tasks			
114	*/			
115	/*			
116	** spin of the tasks			
117	*/			
118	/*			
119	** spin of the tasks			
120	*/			

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/hex.c

DATE 5/23/89
TIME 4:41:14 pm

PAGE #
2/10

LINE #	SOURCE TEXT
121	/*
122	sc_tcreate(diags,
123	PATTERN_TASK_ID, DIAG_PATTERN_TASK_PRI, &err);
124	if(err) output_routine("\nError creating diagnostics task.");
125	*/
126	
127	sc_tcreate(temp_receive_task,
128	RECEIVE_TASK_ID, DIAG_RECEIVE_TASK_PRI, &err);
129	if(err) output_routine("\nError creating receive task.");
130	/* End of main task. main task commits suicide*/
131	sc_delete(0, 0, &err);
132	/* end main */
133	/*
134	/* This is temporarily the rec. task
135	/* to spin off the diag task.
136	*/
137	static void
138	temp_receive_task()
139	{
140	int err;
141	void receive_task();
142	sc_tcreate(diags,
143	PATTERN_TASK_ID, DIAG_PATTERN_TASK_PRI, &err);
144	if(err) output_routine("\nError creating diagnostics task.");
145	receive_task();
146	}
147	/*
148	*****
149	*/
150	sys_out
151	/* Sends message to serial channel A
152	*/
153	*****
154	*/
155	sys_out(buf)
156	{
157	char *buf;
158	while(*buf)
159	{
160	sc_putc(*buf++);
161	}
162	} /* end sys_out */

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/intr.c	DATE 5/23/89	PAGE # 1/11
LINE #		SOURCE TEXT		
1		/* SDCS_ID: intr.c rev 3.1, 4/24/89 at 07:48:34 */		
2		#include "common.h"		
3		#include "in_diags.h"		
4		#include "diag_setjmp.h"		
5		#include "cpu.h"		
6		jmp_buf diag_jmpbuf;		
7		u_long Diag_bus_err_addr = 0;		
8		intr()		
9		{		
10		char board[80];		
11		if (Diag_bus_err_addr != 0) {		
12		map_addr_to_board(Diag_bus_err_addr, board);		
13		(void) send_line(LM_DIAG_ERROR, "\n[BUS ERROR at %08X]\n",		
14		Diag_bus_err_addr, board);		
15		Diag_bus_err_addr = 0;		
16		}		
17		diag_longjmp(diag_jmpbuf, 1);		
18		}		
19		/* grossly simplified cpu memory map */		
20		map_addr_to_board(addr, s)		
21		unsigned long addr;		
22		char *s;		
23		{		
24		char lane;		
25		unsigned long laneoffset;		
26		int bank, pel;		
27		/*		
28		if ((addr >= 0x80000000) && (addr < 0xc0000000)) {		
29		lane = ((addr >> 28) & 7) + 'A';		
30		laneoffset = addr & 0x0ffff;		
31		if (laneoffset < 0x0d000000) {		
32		/* pacs and pams */		
33		if (laneoffset < 0x0c000000) {		
34		bank = (laneoffset >> 26) & 3;		
35		sprintf(s, "(LANE %c PAM BANK %d)", lane, bank);		
36		} else if (laneoffset < 0x0c020000) {		
37		sprintf(s, "(LANE %c PAC LINK TABLE)", lane);		
38		} else {		
39		sprintf(s, "(LANE %c PAC)", lane);		
40		}		
41		} else if (laneoffset < 0x0e000000) {		
42		/* pel's */		
43		pel = (laneoffset >> 12) & 0xf;		
44		sprintf(s, "(LANE %c PEL %d)", lane, pel);		
45		} else if (laneoffset < 0x0f000000) {		
46		strcpy(s, "(TIMING GENERATOR)");		
47		}		
48		} else if ((addr >= CPU_RAM) && (addr < CPU_RAM + CPU_RAM_SIZE)) {		
49		strcpy(s, "(CPU RAM)");		
50		}		

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/lm_diags.c

DATE 5/23/89

PAGE #

TIME 4:41:14 pm

1/12

LINE # SOURCE TEXT

```

1  /* SCCS_ID: lm_diags.c rev 3.3, 5/9/89 at 17:46:40 */
2
3  #include <stdio.h>
4  #include <ctype.h>
5  #include <varargs.h>
6  #include <strings.h>
7  #include "common.h"
8  #include "message.h"
9  #include "network.h"
10 #include "lm_diags.h"
11 #include "lm_xfi.h"
12 #include "fifo.h"
13 #include "task.h"
14 #include "vtr.h"
15 #include "cpu.h"
16 #include "diag_setjmp.h"
17 #include "mod_err.h"
18 #include "svars.h"
19 #include "lm_rd_wr.h"
20
21 #ifdef DIAGS
22 #define DIAGS
23 #endif DIAGS
24 #include "intr.h"
25
26 #ifdef MODELER
27 #include <signal.h>
28 #endif
29
30 #ifdef CPU_DIAGS
31 #define get_key cgetc
32 #define sc_getc cgetc
33 #define sc_putc cputc
34 #define printf cprintf
35 #define check_key cavailable
36 extern char cgetc();
37 extern void cprintf();
38 extern u_long cavailable();
39 int enable_flashing_led = FALSE;
40 #else CPU_DIAGS
41 extern CONNECTION *table_of_conns[];
42 long fifo_task = 0;
43 static char *net_get_line();
44 static char net_get_key();
45 #endif CPU_DIAGS
46
47 static char serial_get_key();
48
49 int new_summary_data;
50 int lm_acceptance = LM_FALSE;
51 int lm_execute_all = LM_FALSE;
52 static int lm_pop_up = LM_FALSE;
53
54 /* Command line modifiable Parameters */
55
56 long lm_current_depth = 0;
57 long lm_start_depth = 0;
58 long lm_fast_test = LM_FALSE;
59
60 static long max_cycles;
61 static long max_errors;
62 static long max_failures;
63 static long max_warnings;
64 static long max_messages;
65 static long summary_count;
66 static long forever;
67
68 static int errors_on;
69 static int warnings_on;
70 static int messages_on;
71 int banner_on;
72
73 static int print_depth;
74
75 char *new_get_parameter();
76
77 #define is_space(c) (((c) == ' ') || ((c) == '\t'))
78 #define is_newline(c) (((c) == '\n') || ((c) == '\r'))
79 #define is_ident(c) ((is_space(c) || ((c) == '-') || ((c) == '\0'))
80 #define max(d1,d2) ((d1) > (d2)) ? (d1) : (d2)
81
82 /* Global counters */
83
84 long total_cycles[MAX_DEPTH];
85 long total_tests[MAX_DEPTH];
86 long total_errors[MAX_DEPTH];
87 long total_error_msgs[MAX_DEPTH];
88 long total_warnings[MAX_DEPTH];
89 long total_messages[MAX_DEPTH];
90 long total_failures;
91
92 long menu_debug = 0;
93 long lm_wizard_mode = 0;
94
95 #ifdef CPU_DIAGS
96 #define LM_PEEK_CHAR(X,O) \
97 ((u_char *) (X) -> incoming_buffer_pointer)[O]
98 #endif VMS
99 #define LM_PEEK_LONG(X) \
100 ( LM_PEEK_CHAR(X,0) | (LM_PEEK_CHAR(X,1) << 8) | \
101 (LM_PEEK_CHAR(X,2) << 16) | \
102 (LM_PEEK_CHAR(X,3) << 24))
103
104 #else SUM4
105 #define LM_PEEK_LONG(X) \
106 ( LM_PEEK_CHAR(X,0) << 24 | \
107 (LM_PEEK_CHAR(X,1) << 16) | \
108 (LM_PEEK_CHAR(X,2) << 8) | \
109 LM_PEEK_CHAR(X,3) )
110 #endif
111 #define LM_PEEK_LONG(X) \
112 *((long *) (X) -> incoming_buffer_pointer)
113 #endif
114
115 static char fifo_user;
116
117 lm_select_mode()
118 {
119     static int saved = FALSE;
120     struct fifo_entry fifo;

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/lm_diags.c	DATE 5/23/89	PAGE # 2/13
LINE #	SOURCE TEXT			
121	CONNECTION *conn;			
122	int size, cmd;			
123				
124	if (saved == TRUE) return;			
125	printf("HIT ANY KEY TO CONTINUE ");			
126	modemprintline("HIT ANY KEY TO CONTINUE ");			
127	while (1) {			
128	if (fifo.no == RX_FIFO;			
129	if (fifo.get(&fifo) == SUCCESS) {			
130	switch (fifo.task) {			
131	case RECEIVE_TASK_ID:			
132	conn = table_of_conns(fifo.user);			
133	cmd = LM_RECV_LONG(conn);			
134	if (cmd != LM_DIAG_GIMGE) {			
135	send_error_message(conn,			
136	"Waiting to begin diagnostics");			
137	set_close_connection_for_server(conn);			
138	continue;			
139	} else {			
140	LM_CHK_PUT_LONG(conn, LM_DIAG_ITEM);			
141	LM_CHK_PUT_LONG(conn, 4);			
142	lm_send_reply(conn);			
143	}			
144	break;			
145	case SERIAL_TASK_ID:			
146	break;			
147	default:			
148	continue;			
149	}			
150	fifo.user = fifo.user;			
151	fifo.task = fifo.task;			
152	saved = TRUE;			
153	return;			
154	}			
155	}			
156				
157	#endif CPU_DIAGS			
158				
159	lm_diag_init()			
160	{			
161	u_long err;			
162				
163	lm_current_depth = 0;			
164	clear_fault_led();			
165	set_global_defaults();			
166	(void) lm_reset_counters((LM_DIAG_MENU * 0));			
167	#ifndef CPU_DIAGS			
168	install_diag_bas_error();			
169	#endif CPU_DIAGS			
170	}			
171				
172	/*			
173	lm_display_menu() returns FAILURE if executed by an all test			
174	and any of the tests is its menu or submenu fails.			
175	lm_display_menu() returns SUCCESS from "quit" or "main".			
176	*/			
177				
178	lm_display_menu(menu)			
179	LM_DIAG_MENU *menu;			
180	{			
181	long return_status;			
182	char string_reply[LM_STR];			
183				
184	if (lm_current_depth == 0) (void) lm_diag_init();			
185	++lm_current_depth;			
186				
187	if ((lm_execute_all == LM_TRUE) (lm_acceptance == LM_TRUE)) {			
188	return_status = execute_all(menu);			
189	--lm_current_depth;			
190	return(return_status);			
191	}			
192				
193	while (LM_TRUE) {			
194	if (lm_pop_up == LM_TRUE) {			
195	if (lm_current_depth > 1) {			
196	--lm_current_depth;			
197	return(SUCCESS);			
198	} else {			
199	lm_pop_up = LM_FALSE;			
200	}			
201	print_menu(menu);			
202	(void) lm_reset_counters(menu);			
203				
204	get_selection(string_reply);			
205	lowerize(string_reply);			
206				
207	lm_start_depth = lm_current_depth;			
208				
209	if (SUCCESS != execute_string(string_reply, menu)) {			
210	lm_current_depth--;			
211	return(SUCCESS);			
212	}			
213	}			
214	}			
215				
216				
217	run_cont(menu)			
218	LM_DIAG_MENU *menu;			
219	{			
220	long count;			
221				
222	count = 0;			
223				
224	while (SUCCESS != lm_check_key()) {			
225	execute_routine(menu, (long) 1);			
226	if (SUCCESS != failure_count_check()) {			
227	/* send_line(LM_DIAG_WHITE, "[P1]\n"); */			
228	print_summary(menu);			
229	break;			
230	}			
231	if (summary_count != 0) {			
232	count++;			
233	if (0 == ((count) & summary_count)) {			
234	/* send_line(LM_DIAG_WHITE, "[P2]\n"); */			
235	print_summary(menu);			
236	count = 0;			
237	}			
238	}			
239				
240	}			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/lm_diags.c	DATE 5/23/89	PAGE # 3/14
LINE #	SOURCE TEXT			
241	lm_get_input(prompt, reply, lno)			
242	char *prompt;			
243	char *reply;			
244	int lno;			
245	{			
246	(void) send_line(LM_DIAG_NULL, "%s", prompt);			
247	lm_get_line(reply, lno);			
248	}			
249	}			
250	print_menu(menu)			
251	register LM_DIAG_MENU *menu;			
252	{			
253	register LM_DIAG_MENU_ITEM *menu_list;			
254	register long i;			
255	if (lm_wizard_mode)			
256	print_menu_title(menu->title);			
257	else			
258	(void) send_line(LM_DIAG_ITEM, "%s\n", menu->title);			
259	/* don't display "e" unless there is a test or test menu is here */			
260	menu_list = menu->menu_items;			
261	for (i=0; i<menu->number_of_items; i++, menu_list++) {			
262	if (!((menu_list->attributes & LM_DIAG_no_execute))) {			
263	send_line(LM_DIAG_WHITE, "%s", menu_list->name);			
264	break;			
265	}			
266	}			
267	menu_list = menu->menu_items;			
268	for (i=0; i<menu->number_of_items; i++, menu_list++) {			
269	if (!((lm_wizard_mode			
270	&& (menu_list->attributes & LM_DIAG_wizard_mode)))			
271	continue;			
272	if (!((menu_list->attributes & LM_DIAG_no_display)))			
273	(void) send_line(LM_DIAG_ITEM, "%s", menu_list->name);			
274	(menu_list->attributes & LM_DIAG_no_select)?			
275	"%s: menu_list->selection,			
276	menu_list->menu_text);			
277	}			
278	send_line(LM_DIAG_WHITE, "%s", menu_list->name);			
279	"			
280	send_line(LM_DIAG_WHITE, "%s", menu_list->name);			
281	"			
282	send_line(LM_DIAG_WHITE, "%s", menu_list->name);			
283	"			
284	send_line(LM_DIAG_WHITE, "%s", menu_list->name);			
285	"			
286	}			
287	print_menu_title(a)			
288	register char *a;			
289	{			
290	char title[256];			
291	register char *t;			
292	register long count, i, line;			
293	{			
294	(void) send_line(LM_DIAG_WHITE, "\n");			
295	while (*a == ' ')			
296	++a;			
297	for (line = 0, count = strlen(a); ++line) {			
298	t = title;			
299	if (count > 32) {			
300	for (count = 32; count && (a[count] != ' '); --count;			
301	if (count)			
302	count = 32;			
303	/* one tab on the first line */			
304	*t++ = '\t';			
305	/* two tabs on all continuation lines */			

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/lm_diags.c	DATE 5/23/89	PAGE # 4/15
LINE #	SOURCE TEXT			
361	read_line(LM_DIAG_WHITE, "\n");			
362	New_summary_data = FALSE;			
363	}			
364	}			
365	executeRoutine(memu, count)			
366	LM_DIAG_MESS = memu;			
367	long count;			
368	{			
369	long loop;			
370	register long return_status, i;			
371	register long no_loading_banner;			
372	register int repeat_banner;			
373	register char *s;			
374	char banner_string[256], dots[50];			
375	long attr;			
376	{			
377	(void) lm_check_key(); /* is there an interrupt? */			
378	attr = memu->memu_items[memu->current_selection].attributes;			
379	{			
380	/* Don't execute routines which have the no_execute bit set			
381	and we are doing an 'all test' or acceptance test */			
382	if ((lm_execute_all lm_acceptance) && (attr & LM_DIAG_no_execute)) {			
383	return SUCCESS;			
384	}			
385	/* Don't execute routines which don't have the acceptance bit set			
386	and we are doing an 'acceptance test' */			
387	if (lm_acceptance && !(attr & LM_DIAG_acceptance)) {			
388	return SUCCESS;			
389	}			
390	{			
391	repeat_banner = (messages_on warnings_on errors_on);			
392	no_loading_banner = (attr & LM_DIAG_memo_banner)			
393	&& !(lm_execute_all lm_acceptance);			
394	sprintf(banner_string, "%s",			
395	(lm_current_depth - lm_start_depth) * 2, "");			
396	memu->memu_items[memu->current_selection].memo_text);			
397	{			
398	if (attr & LM_DIAG_no_repeat) {			
399	count = 1;			
400	}			
401	TEST_RUN(memu);			
402	for (loop = 0; loop < count; loop++) {			
403	(void) lm_check_key(); /* is there an interrupt? */			
404	if (SUCCESS != failure_count_check()) {			
405	break;			
406	}			
407	if (banner_on && !(attr & LM_DIAG_no_banner) && !no_loading_banner) {			
408	lm_force_start_of_line(LM_DIAG_MESS);			
409	lm_banner("%s", banner_string,			
410	repeat_banner (attr & LM_DIAG_memo_banner)) ? "\n" : "...");			
411	}			
412	return_status			
413	= (memu->memu_items[memu->current_selection].action_routine)			
414	(memu, memu->memu_items[memu->current_selection].user_data);			
415	if (attr & LM_DIAG_log_results) {			
416	++total_tests(lm_current_depth);			
417	New_summary_data = TRUE;			
418	}			
419	if (return_status != SUCCESS) {			
420	test_failed(memu);			
421	if (attr & LM_DIAG_log_results) {			
422	set_fault_led();			
423	++total_errors(lm_current_depth);			
424	++total_failures;			
425	#ifdef CPU_DIAGS			
426	if (Pifo_task == RECEIVE_TASK_ID) {			
427	set_report_failure();			
428	}			
429	#endif CPU_DIAGS			
430	{			
431	if (!(attr & LM_DIAG_memo_banner)) {			
432	if (((return_status != SUCCESS) && (errors_on == LM_TRUE))			
433	(banner_on && !(attr & LM_DIAG_no_banner))) {			
434	if (repeat_banner) {			
435	lm_force_start_of_line(LM_DIAG_MESS);			
436	lm_banner("%s...", banner_string);			
437	}			
438	for (s = dots, i = strlen(banner_string); i < 67; ++i) *s++ = '.';			
439	(void) strcpy(s, (return_status == SUCCESS) ? "PASS\n" :			
440	"....FAIL\n");			
441	lm_banner(dots);			
442	}			
443	} else if ((return_status != SUCCESS) && (errors_on == LM_TRUE)			
444	&& (lm_execute_all lm_acceptance)) {			
445	lm_force_start_of_line(LM_DIAG_MESS);			
446	lm_banner("%s produced failures.\n", banner_string);			
447	}			
448	if (!(lm_execute_all lm_acceptance forever)) {			
449	if ((summary_count != 0) && (0 == ((loop+1) % summary_count))) {			
450	/* send_line(LM_DIAG_WHITE, "[PS]\n"); */			
451	print_summary(memu);			
452	}			
453	}			
454	return return_status;			
455	}			
456	#ifdef CPU_DIAGS			
457	flash_diag_led()			
458	{			
459	register u_long delay_count;			
460	delay_milliseconds(delay_count, 500);			
461	((cpu_control_reg_struct *)CPU_CONTROL_REG)->not_test_led = not_LED_OFF;			
462	delay_milliseconds(delay_count, 500);			
463	((cpu_control_reg_struct *)CPU_CONTROL_REG)->not_test_led = not_LED_ON;			
464	}			
465	#endif CPU_DIAGS			
466	failure_count_check()			
467	{			
468	return ((max_failures > 0) && (total_failures >= max_failures))?			
469	FAILURE: SUCCESS;			
470	}			
471	}			

Copyright 1989

Logic Modeling Systems

SOURCE PROGRAM

diags/lm_diags.c

DATE 5/23/89

PAGE #

TIME 4:41:14 pm

5/16

```

LINE # SOURCE TEXT
481 error_count_check()
482 {
483     if (((max_errors>0) && (total_error_msg(lm_current_depth) > max_errors))
484         || ((max_warnings>0) && (total_warnings(lm_current_depth) > max_warnings))
485         || ((max_messages>0) && (total_messages(lm_curr...depth) > max_messages)))
486         return(FAILURE);
487     return(SUCCESS);
488 }
489
490 static struct help_list {
491     int print_always;
492     char *text;
493 } help_list[] = {
494     1, "Built in Commands: \n",
495     1, "\t all          (a)          execute all tests\n",
496     1, "\t quit          (q)          exit from this menu\n",
497     1, "\t main          (m)          goto the main menu\n",
498     1, "\t exit          (e)          exit from this program\n",
499     1, "\t help          (h)          display this information\n",
500     1, "\t \n",
501     1, "Command Qualifiers: \n",
502     1, "\t continuous    (c) = <t/f> run test continuously\n",
503     1, "\t repeat        (r) = <n> run test <n> times\n",
504     1, "\t summary       (s) = <n> print summary every <n> cycles\n",
505     0, "\t error_count   (e) = <n> set error threshold to <n> errors\n",
506     0, "\t warning_count (w) = <n> set warning threshold to <n> warnings\n",
507     0, "\t message_count (m) = <n> set message threshold to <n> messages\n",
508     0, "\t failure_count (f) = <n> stop testing after <n> tests fail\n",
509     0, "\t fast_test     (fast) = <t/f> run short version of test if available\n",
510     0, "\t \n",
511     0, "Output Qualifiers: \n",
512     0, "\t print_errors  (pe) = <t/f> suppresses error messages\n",
513     0, "\t print_warnings (pw) = <t/f> suppresses warning messages\n",
514     0, "\t print_messages (pm) = <t/f> suppresses messages\n",
515     0, "\t print_banners (pb) = <t/f> suppresses test banners\n",
516     1, "\t \n",
517     1, " \n",
518 #ifdef MODELER
519     0, "\t output        (o) = <filename> log output to the named file\n",
520     0, "\t \n",
521 #endif
522     0, 0
523 };
524
525 help()
526 {
527     print_text_array(help_list);
528     (void) send_line(LM_DIAG_NULL, "hit return to continue");
529     (void) lm_get_key();
530 }
531
532 print_text_array(help_list);
533 struct help_list *help_list;
534 {
535     for (; help_list->text; ++help_list)
536         if (lm_wizard_mode || help_list->print_always)
537             (void) send_line(LM_DIAG_HELP, help_list->text);
538 }
539
540 /*
541 ** execute_all() returns FAILURE if any of the tests in
542 ** its menu or submenu fails.
543 **
544 */
545 execute_all(menu)
546     LM_DIAG_MENU *menu;
547 {
548     long i; return_value = SUCCESS;
549     for (i=0; i<menu->number_of_items; ++i) {
550         if (SUCCESS != failure_count_check()) {
551             break;
552         }
553         menu->current_selection = i;
554         if (execute_routine(menu, (long) i) != SUCCESS)
555             return_value = FAILURE;
556     }
557 #ifdef CPU_DIAGS
558     /*
559     * CPU diagnostics enter at depth=1, and immediately
560     * increment to 2 in the top level menu, so flash the
561     * LED only in between tests at the top level menu.
562     */
563     if ((enable_flashing_led) && (lm_current_depth == 2) &&
564         (! (menu->menu_items[menu->current_selection].attributes &
565            LM_DIAG_no_execute)))
566         flash_diag_led();
567 #endif CPU_DIAGS
568     if (menu->menu_items[i].status & LM_DIAG_stop_test)
569         break; /* or should we return? */
570     ++total_cycles(lm_current_depth);
571     return return_value;
572 }
573
574 lm_reset_counters(menu)
575     LM_DIAG_MENU *menu;
576 {
577     long i;
578     LM_DIAG_MENU_ITEM *menu_item;
579     New_summary_data = FALSE;
580     for (i=0; i<MAX_DEPTH; i++) {
581         total_cycles[i] = 0;
582         total_tests[i] = 0;
583         total_errors[i] = 0;
584         total_error_msg[i] = 0;
585         total_messages[i] = 0;
586         total_warnings[i] = 0;
587     }
588     total_failures = 0;
589     if (menu) {
590         menu_item = menu->menu_items;
591         for (i=0; i<menu->number_of_items; i++, menu_item++) {
592             menu_item->status = 0;
593         }
594     }
595     return(SUCCESS);
596 }
597
598
599
600

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/lm_diags.c

DATE 5/23/89
TIME 4:41:14 pm

PAGE #
6/17

```

LINE #          SOURCE TEXT
601 }
602 lm_increment_cycle_count()
603 {
604     ++total_cycles(lm_current_depth);
605     if (total_cycles(lm_current_depth) > max_cycles) return(FAILURE);
606     return(SUCCESS);
607 }
608
609 static int at_start_of_line = 1;
610
611 lm_force_start_of_line(print_mode)
612 {
613     if (!at_start_of_line)
614         (void) send_line(print_mode, "\n", 0);
615     at_start_of_line = 1;
616 }
617
618 /*VARARGS*/ /*ARGUSED*/
619 lm_error(va_alist)
620 {
621     char *va_alist;
622     va_list argp;
623     char lm_diag_buffer[255];
624     char format[255];
625
626     (void) lm_check_key(); /* look for interrupt */
627
628     ++total_error_msgs(lm_current_depth);
629     if ((LM_TRUE == errors_on) && (print_depth > lm_current_depth)) {
630         lm_force_start_of_line(LM_DIAG_ERROR);
631         va_start(argp);
632         strcpy(format, "ERROR: ");
633         strcat(format, va_arg(argp, char *));
634         (void) vfprintf(lm_diag_buffer, format, argp);
635         va_end(argp);
636         (void) do_send_line(LM_DIAG_ERROR, lm_diag_buffer);
637         at_start_of_line
638             = is_newline(lm_diag_buffer(strlen(lm_diag_buffer)-1));
639     }
640     if ((max_errors > 0) &&
641         (total_error_msgs(lm_current_depth) >= max_errors))
642         return(FAILURE);
643     return(SUCCESS);
644 }
645
646 /*VARARGS*/ /*ARGUSED*/
647 lm_warning(va_alist)
648 {
649     char *va_alist;
650     va_list argp;
651     char lm_diag_buffer[255];
652     char format[255];
653
654     (void) lm_check_key(); /* look for interrupt */
655
656     ++total_warnings(lm_current_depth);
657     if ((LM_TRUE == warnings_on) && (print_depth > lm_current_depth)) {
658         lm_force_start_of_line(LM_DIAG_WARN);
659         va_start(argp);
660         strcpy(format, "WARNING: ");
661         strcat(format, va_arg(argp, char *));
662         (void) vfprintf(lm_diag_buffer, format, argp);
663         va_end(argp);
664         (void) do_send_line(LM_DIAG_WARN, lm_diag_buffer);
665         at_start_of_line
666             = is_newline(lm_diag_buffer(strlen(lm_diag_buffer)-1));
667     }
668     if ((max_warnings > 0) &&
669         (total_warnings(lm_current_depth) >= max_warnings))
670         return(FAILURE);
671     return(SUCCESS);
672 }
673
674 /*VARARGS*/ /*ARGUSED*/
675 lm_message(va_alist)
676 {
677     char *va_alist;
678     va_list argp;
679     char lm_diag_buffer[255];
680     char format[255];
681
682     (void) lm_check_key(); /* look for interrupt */
683
684     ++total_messages(lm_current_depth);
685     if ((LM_TRUE == messages_on)
686         && (print_depth > (lm_current_depth - lm_start_depth))) {
687         format = va_arg(argp, char *);
688         (void) vfprintf(lm_diag_buffer, format, argp);
689         va_end(argp);
690         (void) do_send_line(LM_DIAG_MESS, lm_diag_buffer);
691         at_start_of_line
692             = is_newline(lm_diag_buffer(strlen(lm_diag_buffer)-1));
693     }
694     if ((max_messages > 0) &&
695         (total_messages(lm_current_depth) >= max_messages))
696         return(FAILURE);
697     return(SUCCESS);
698 }
699
700 /*VARARGS*/ /*ARGUSED*/
701 /* always print this */
702 lm_banner(va_alist)
703 {
704     char *va_alist;
705     va_list argp;
706     char lm_diag_buffer[255];
707     char format[255];
708
709     (void) lm_check_key(); /* look for interrupt */
710
711     va_start(argp);
712     format = va_arg(argp, char *);
713     (void) vfprintf(lm_diag_buffer, format, argp);
714     va_end(argp);
715     (void) do_send_line(LM_DIAG_MESS, lm_diag_buffer);
716     at_start_of_line
717         = is_newline(lm_diag_buffer(strlen(lm_diag_buffer)-1));
718     return(SUCCESS);
719 }
720

```

Copyright 1989
Logic Modeling Systems

SOURCE PROGRAM
diags/lm_diags.c

DATE 5/23/89 PAGE #
TIME 4:41:14 pm 7/18

```

721 }
722
723 clean_input(line)
724 char *line;
725 { char temp_string[1024]; /* use a temporary string to avoid */
726   char *str_ptr; /* portability problems */
727   (void) strcpy(temp_string, line);
728   str_ptr = temp_string;
729   while (!is_space(str_ptr[0]))
730     str_ptr++; /* remove leading white space */
731   (void) strcpy(line, str_ptr);
732 }
733
734 /*VARARGS2*/
735 /*ARGSUSED*/
736 send_line(line_type, fmt, va_alist)
737 int line_type;
738 char *fmt;
739 char **va_alist;
740 { va_list pvar;
741   char str[MAX_SERVER_PACKET];
742   (void) vsprintf(str, fmt, pvar);
743   do_send_line(line_type, str);
744 }
745
746 do_send_line(line_type, str)
747 char *str;
748 {
749   #ifdef CPU_DIAGS
750   CONNECTION *conn;
751   int bytes;
752   char text[MAX_SERVER_PACKET];
753   u_long cmd;
754   if (Pifo_task == RECEIVE_TASK_ID) {
755     bytes = strlen(str);
756     get_request(conn, &cmd, text);
757     /* Who cares what the cmd was? */
758     LM_CHK_PUT_LONG(conn, line_type);
759     LM_CHK_PUT_LONG(conn, bytes+4);
760     for (i=0; i<bytes; i++) {
761       LM_CHK_PUT_CHAR(conn, str[i]);
762     }
763     lm_send_reply(conn);
764   } else {
765     if (Pifo_user == 0) {
766       printf("%s", str);
767     } else {
768       modemprintline(str);
769     }
770   }
771   #else CPU_DIAGS
772   printf("%s", str);
773   #endif CPU_DIAGS
774 }
775
776 exit_request()
777 {
778   /* modeler requesting death */
779   #ifdef CPU_DIAGS
780   (void) send_line(LM_DIAG_END, "");
781   #else CPU_DIAGS
782   u_long err;
783   char skip_menu_val = CPU_SKIP_MENU_YES;
784   (void) lm_svarm_access((char *) 0, 0, 0, MEMORY_DIAG_INIT, &err);
785   (void) lm_svarm_access(&skip_menu_val, CPU_SKIP_MENU,
786     SILENCE_CPU_SKIP_MENU, MEMORY_WRITE, &err);
787   #endif CPU_DIAGS
788   lm_reset_cpu();
789 }
790
791 lm_reset_cpu()
792 {
793   #ifdef MODELER
794   register cpu_control_reg_struct *control_reg;
795   register u_long delay;
796   /* send a reply to the hardware that we're outta here */
797   control_reg = (cpu_control_reg_struct *) CPU_CONTROL_REG;
798   control_reg->suicide = 1;
799   delay_milliseconds(delay, 8000);
800   #else MODELER
801   kill(getpid(), SIGTERM);
802   #endif MODELER
803 }
804
805 char
806 lm_get_key()
807 { register char c;
808   do {
809     #ifdef CPU_DIAGS
810     if (Pifo_task == RECEIVE_TASK_ID) {
811       c = net_get_key();
812     } else {
813       c = serial_get_key();
814     }
815     #else CPU_DIAGS
816     c = get_key();
817     #endif CPU_DIAGS
818     if (c == INTR_CHARACTER) intr();
819   } while (c == ctrl(s)); /* simple 'S' strategy:
820     any char resumes printing */
821   return c;
822 }
823
824 static char
825 serial_get_key()
826 {
827   #ifdef CPU_DIAGS
828

```

Copyright 1989 Logic Modeling Systems		SOURCE PROGRAM diags/lm_diags.c	DATE 5/23/89	PAGE # 8/19
LINE #	SOURCE TEXT			
841	CONNECTION *conn;			
842	u_long cmd;			
843	char buffer[MAX_SERVER_PACKET];			
844	get_r_socket(&conn, &cmd, buffer);			
845	return far(0);			
846	#else CPU_DIAGS			
847	char c;			
848	c = get_key();			
849	#endif CPU_DIAGS			
850	#endif CPU_DIAGS			
851	static char *			
852	serial_get_line(string, len)			
853	register char *string;			
854	{			
855	register char *s = string, *limit = s + len;			
856	char c;			
857	static char backspace[] = "\b \b";			
858	while ((c = lm_get_key()) != CR) && (c != LF) && (s < limit) {			
859	switch (c) {			
860	case BS:			
861	case DEL:			
862	if (s > string) {			
863	--s;			
864	serial_puts(backspace);			
865	break;			
866	case ctrl(u):			
867	while (s > string) {			
868	--s;			
869	serial_puts(backspace);			
870	break;			
871	case ctrl(w):			
872	while ((s > string) && isspace(s[-1])) {			
873	--s;			
874	serial_puts(backspace);			
875	break;			
876	case ctrl(v):			
877	while ((s > string) && isspace(s[-1])) {			
878	--s;			
879	serial_puts(backspace);			
880	break;			
881	default:			
882	if (isprint(c)) {			
883	*s++ = c;			
884	serial_puts(c);			
885	} else {			
886	serial_puts(c);			
887	serial_puts(ctrl(g));			
888	}			
889	}			
890	serial_puts('\n');			
891	*s = '\0';			
892	return string;			
893	}			
894	char *			
895	lm_get_line(reply, len)			
896	register char *reply;			
897	long len;			
898	{			
899	register char *c;			
900	#ifndef CPU_DIAGS			
901	if (Fifo_task == RECEIVE_TASK_ID) {			
902	c = net_get_line(reply, len);			
903	} else			
904	#endif CPU_DIAGS			
905	{			
906	c = serial_get_line(reply, len);			
907	while (*reply)			
908	if (*reply++ == INT_CHARACTER) istr();			
909	return c;			
910	}			
911	serial_puts(str)			
912	char *str;			
913	{			
914	#ifndef CPU_DIAGS			
915	if (Fifo_user == 0) printf("ts", str);			
916	else modmodprintline(str);			
917	#else CPU_DIAGS			
918	printf("ts", str);			
919	#endif CPU_DIAGS			
920	}			
921	serial_puts(c)			
922	{			
923	#ifndef CPU_DIAGS			
924	if (Fifo_user == 0) printf("tc", c);			
925	else putmodemc(c);			
926	#else CPU_DIAGS			
927	printf("tc", c);			
928	#endif CPU_DIAGS			
929	}			
930	get_selection(string_reply)			
931	char *string_reply;			
932	{			
933	#ifndef CPU_DIAGS			
934	if (Fifo_task == RECEIVE_TASK_ID) net_get_selection(string_reply);			
935	else			
936	#endif CPU_DIAGS			
937	{			
938	(void) send_line(LM_DIAG_WHITE, "\n");			
939	do {			
940	lm_get_input("selection: ", string_reply, max_str);			
941	clear_input(string_reply);			
942	} while (strlen(string_reply) == 0);			
943	(void) send_line(LM_DIAG_WHITE, "\n");			
944	}			
945	/* lm_check_key flushes the input buffer */			